UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
FADEL TOURE

INDICATEUR DE QUALITÉ POUR LES SYSTÈMES ORIENTÉS OBJET : VERS UN MODÈLE UNIFIANT PLUSIEURS MÉTRIQUES

Décembre 2007

INDICATEUR DE QUALITÉ POUR LES SYSTÈMES ORIENTÉS OBJET : VERS UN MODÈLE UNIFIANT PLUSIEURS MÉTRIQUES

Fadel TOURE

SOMMAIRE

Les systèmes industriels actuels sont de plus en plus complexes. La gestion (assurance et contrôle) de leur qualité répond à des enjeux techniques et économiques importants. La qualité des logiciels est un concept complexe. Elle possède plusieurs caractéristiques. Les métriques, dans ce contexte, constituent de véritables « sondes » permettant d'évaluer plusieurs attributs de la qualité.

Pour auditer la qualité d'un logiciel, la métrologie, comme dans tous les domaines d'ingénierie, reste le moyen le plus objectif et le plus fiable. Le domaine des métriques, en particulier orientées objet, a fait l'objet d'une multitude de travaux. Plusieurs de ces métriques ont été largement expérimentées et discutées dans la littérature. Par contre, certaines d'entre elles n'ont pas fait l'objet de grandes validations. Par ailleurs, leur nombre, assez important, surtout dans le cas des systèmes orientés objet, soulève plusieurs problèmes, en particulier, la quantité d'information qu'elles fournissent qui est difficile à assimiler et à gérer (voire même à interpréter), d'une part, et les recoupements qui existent entre plusieurs d'entre elles, d'autre part.

L'idée d'unifier les métriques pour capturer les attributs de haut niveau est la principale motivation de ce travail. Le modèle que nous proposons (indicateurs de qualité Qi), dans ce mémoire, est basé sur les graphes de contrôle, les probabilités d'invocation de modules, leur complexité cyclomatique et les taux de couverture de test. Son expérimentation, sur plusieurs projets logiciels d'envergure, nous a permis de démontrer son efficacité, à plusieurs niveaux, relativement à plusieurs métriques orientées objet existantes. Les résultats obtenus démontrent que notre modèle est un bon indicateur de plusieurs attributs qualité tels que la changeabilité et la testabilité sans compter certains attributs capturés de façon indirecte. Nous avons aussi développé un outil supportant ce modèle, permettant de déterminer les indicateurs de qualité de chaque composante d'un système logiciel écrit en Java.

INDICATOR OF QUALITY FOR OBJECT ORIENTED SYSTEMS: TOWARDS A MODEL UNIFYING SEVERAL METRIC

Fadel TOURE

ABSTRACT

Current industrial systems are increasingly complex. The management (insurance and control) of their quality answers important technical and economic stakes. The quality of software is a complex concept. It has several characteristics. Metrics, in this context, constitute genuine "probes" making it possible to evaluate several attributes of quality.

To probe the quality of a software, metrology, as in all the fields of engineering, remains the most objective means and most reliable. The field of metrics, in particular in the object oriented paradigm, was the subject of a multitude of proposals.

Several of these metrics were largely tested and discussed in the literature. On the other hand, some of them were not the subject of great validations. In addition, their number, rather significant, especially in the case of object-oriented systems, raises several problems, in particular, the quantity of information which they provide which is difficult to assimilate and manage (and even to interpret), on the one hand, and the overlapping which exist between several of them, on the other hand.

The idea to unify some metrics to capture high level attributes is the principal motivation of this work. The model that we propose (Quality Indicator Qi), in this work, is based on control graphs, probabilities of methods invocation, their cyclomatic complexity and their testing coverage. Its experimentation, on several large-scale software projects, enabled us to show his effectiveness, on several levels, relatively to several existing object-oriented metrics. The results obtained show that our model is a good indicator of several quality attributes such as the changeability and testability without counting certain attributes captured in an indirect way. We also developed a tool supporting this model, allowing to determine the indicators of quality of each component of a software system written in Java.

REMERCIEMENTS

Merci,

A Dieu de m'avoir donné la volonté et de m'avoir maintenu dans des conditions de santé excellente durant ce travail.

A mes professeurs Mourad Badri et Linda BADRI, pour leur soutient intellectuel et financier, leur disponibilité et leur encadrement sans faille. Merci pour ces plaisants moments passés à amasser des connaissances. Votre patience avec les étudiants, votre volonté pour le travail, et votre passion pour le génie logiciel m'ont beaucoup inspiré pour mener à terme ce mémoire.

A mon père et ma mère, pour leur soutient moral et matériel constant, leur prières et leur souhait qui m'ont été d'une aide incommensurable.

A mes frères, mes sœurs et mes proches parents qui m'ont assisté et encouragé durant toutes mes études malgré la distance qui nous sépare.

A mes amis pour la disponibilité dont ils ont fait preuve, le réconfort qu'ils m'ont apporté lors des moments difficiles, merci d'avoir été la pour moi lorsque le moral était au plus bas.

Cette étude a été rendue possible grâce à la contribution financière du CRSNG (Conseil de Recherches en Sciences Naturelles et en Génie du Canada) et à la fondation de l'UQTR.

TABLE DE MATIERES

Pages

LISTE DES TABLEAUX

LISTE DES FIGURES

Pages

CHAPITRE 1

INTRODUCTION

Les systèmes logiciels développés de nos jours sont de plus en plus volumineux et complexes. Les exigences qualité ont gagné en importance au niveau technique, en particulier lorsque le logiciel a un usage critique, et économique car les frais importants engagés par l'achat (ou développement) d'un logiciel méritent en retour un certain standard de qualité. C'est ainsi que le concept de qualité s'est développé avec le temps, regroupant plusieurs caractéristiques, sous-caractéristiques, et attributs internes.

Trouver, dans ce contexte, une manière adéquate et efficace pour mesurer la qualité pour pouvoir la gérer, surtout dans le cas des systèmes complexes (taille et structure), reste une gymnastique complexe et difficile. Nous nous intéressons dans ce mémoire au paradigme orienté objet qui, en quelques décennies, a fait ses preuves en matière de développement de logiciels de qualité (apport important) et qui tend à devenir l'approche standard dans la programmation. Les systèmes orientés objet possèdent plusieurs spécificités. Leurs composantes, de part leur nature, présentent de multiples dépendances. À cet effet, Chidamber et Kemmerer [Chidamber 94] ont été les premiers à introduire des métriques orientées objet au début des années 90. Leurs premiers travaux de recherche sur le sujet, réalisés au prestigieux MIT, ainsi que l'introduction de nouvelles métriques par la suite, grâce aux travaux de plusieurs chercheurs dans le domaine, a permis de mettre en place les premiers éléments d'un cadre structuré définissant les attributs de qualité des systèmes orientés objet et comment aborder leur évaluation.

Plusieurs travaux portant sur les métriques orientées objet ont, en fait, été réalisés depuis le début des années 90, notamment au niveau de la définition, de

la validation, de l'évaluation, ou encore du raffinement. C'est un sujet qui a attiré (et continue jusqu'à nos jours) de nombreux chercheurs. La plupart de ces métriques ont été largement expérimentées et discutées dans la littérature. Certaines d'entre elles ont été validées et d'autres restent discutables.

1.1 Problématique

La plupart des métriques orientées objet définies à ce jour capturent des attributs de qualité internes (bas niveau) tels que la complexité, le couplage, la cohésion, la taille, etc. Pour capturer des attributs de haut niveau, des combinaisons de ces métriques deviennent indispensables. Cependant, la validation de ces métriques pour la capture d'attributs de qualité complexes comme la maintenabilité, la fiabilité, la réutilisabilité, ou encore la testabilité reste une tâche très complexe et difficile; et cela, d'autant plus que, des études récentes [Aggarwal 06] montrent que beaucoup de ces métriques se recoupent d'une part et d'autre part, certaines d'entre elles capturent mal les attributs de qualité des systèmes logiciels, car biaisées par la taille des systèmes qui constitue souvent un facteur de confusion tel que mentionné dans [El Emam 99]. Comment, dans ce cadre, mettre en place une métrique (ou noyau de quelques métriques) capable de cibler certains attributs bien précis de la qualité de haut niveau, en regroupant le maximum de métriques concourantes ?

1.2 Approche

Ce mémoire s'inscrit dans ce cadre et présente essentiellement les résultats de diverses expérimentations sur des projets d'envergure d'un nouveau modèle unifiant plusieurs métriques orientées objet : les « Indicateurs de Qualité » (Qi). Le modèle original est tiré des travaux antérieurs de Badri & al. [Badri 95, Badri 96, Badri 99] et récents [Badri 05]. Ce modèle a déjà fait l'objet de

plusieurs travaux antérieurs (et publications), essentiellement, dans le contexte du test des systèmes orientés objet et de l'analyse de l'impact des modifications (éléments de base du modèle – graphes de contrôle réduits aux appels). Ces intérêts majeurs étaient, entre autres, l'orientation du processus de test et en particulier celui de l'intégration des classes et des tests de régression (dans une optique de maintenance également). Ce modèle, intégrant plusieurs dimensions reliées à la qualité, pouvait aussi servir à prédire plusieurs attributs qualité. Ces différents éléments n'ont jamais été investigués (de façon empirique assez large). Il s'agissait donc dans ce projet de recherche, essentiellement, d'affiner son adaptation aux systèmes orientés objet (intégration de spécificités supplémentaires), de procéder à des expérimentations de grande échelle, de l'améliorer éventuellement, suite à ces expérimentations, et de valider les nombreuses hypothèses concernant son utilisation comme indicateur de plusieurs attributs de qualité (hypothèses discutées ultérieurement).

Le modèle en question, tel qu'il a été conçu et en fonction des paramètres qu'il considère, intègre en fait, plusieurs dimensions reliées (directement ou indirectement) à plusieurs caractéristiques importantes telles que : le taux de couverture de test des composants, la dépendance de contrôle entre composants, la complexité cyclomatique des composants ainsi que les probabilités d'appel des modules d'un code. Ce dernier point est relié au profil opérationnel des logiciels. Ces différents éléments ont été intégrés dans un seul modèle (modèle original) qui les pondère donc en tenant compte du profil opérationnel des systèmes.

En plus de supporter, à différents niveaux, le processus de test des systèmes orientés objet, nous pensons que les Qi sont également des indicateurs puissants pouvant être utilisés pour renseigner plusieurs attributs de la qualité des systèmes logiciels complexes, entre autres :

- La complexité d'un module est souvent à l'origine des fautes. La probabilité de découvrir ces fautes est liée à l'effort de test appliqué durant le

développement, et à la fréquence d'utilisation du module considéré durant le cycle de vie du logiciel. Le fait de combiner les probabilités d'appel à la complexité cyclomatique dans le modèle devrait donner une bonne prévision des fautes contenues dans les modules ainsi que la probabilité des les découvrir.

- Les changements apportés à un logiciel se justifient, entre autres, par une nécessité d'adaptation ou par la mise au point suite à la découverte de fautes dans ses composants. De ce fait, notre modèle, qui permet déjà de pointer les composants les plus critiques, devrait aussi prédire ceux qui sont les plus susceptibles à subir le changement durant l'évolution d'un logiciel (versions successives). Notons, par ailleurs, que l'évolution des logiciels est aussi le résultat de l'évolution inéluctable des besoins. Cette évolution peut être cependant due à des corrections (mineures ou majeures) ainsi qu'à des adaptations à de nouveaux environnements.

- Les diverses caractéristiques de la « testabilité » d'une classe, qui sont en grande partie liées à son couplage vis-à-vis des autres classes du système et à sa complexité devraient être capturées par notre modèle.

- Notre modèle devrait aussi, de part sa définition et sa construction, pouvoir (de façon directe ou indirecte) capturer les caractéristiques d'un certain nombre de métriques connues et largement utilisées en génie logiciel.

- L'intégration de la complexité et du couplage probabiliste peut faire des Qi un outil d'aide à la décision (encore plus poussée grâce à son affinement) dans l'orientation des tests d'une manière générale (en particulier d'intégration et de régression), de détection de cycles, ainsi que d'analyse de l'impact des changements.

Ces différentes hypothèses sont reliées à des enjeux importants et d'actualité dans le domaine du développement et de la maintenance des logiciels de grande

taille. Les nombreuses expérimentations effectuées devront permettre la confirmation (ou infirmation) de ces hypothèses. Leur confirmation permettrait bien entendu de valider notre modèle (affiné) comme un bon indicateur de qualité, d'une part, et un bon outil pour la maintenance (évolution) des logiciels, d'autre part.

Le modèle a été implémenté dans le cadre d'un environnement complet (composée de plusieurs outils) d'expérimentation que nous avons construit. Plusieurs expérimentations d'envergure ont été conduites sur des systèmes orientés objet (Java) de grande taille. Les résultats obtenus sont très positifs à plusieurs niveaux. Ils permettent également d'envisager plusieurs pistes de recherche dans le cadre de travaux futurs. Plusieurs publications portant sur ce travail ainsi que sur les résultats importants obtenus sont prévues.

1.3 Organisation

Le présent mémoire est divisé en trois grandes parties. Dans un premier temps, et comme le domaine des métriques orientées objet et leur utilisation est très vaste et largement documenté dans la littérature, nous ferons un bref état de l'art dans lequel nous présenterons, essentiellement, un résumé des principales métriques orientées objet (les plus utilisées ont été considérées dans nos expérimentations), ainsi que les principaux travaux effectués récemment sur certaines questions importantes et d'actualité se rapportant aussi bien aux métriques qu'à des questions concernant leur recoupement, leur regroupement, la prédiction des changements, la testabilité et les modèles d'analyse de l'impact de changements. Ceci nous permettra, en plus de dresser un bref bilan des travaux récents dans ce domaine, de présenter les forces et faiblesses des principales propositions. En effet, bien que les premiers travaux concrets dans le domaine des métriques orientées objet remontent aux débuts des années 90, nous observons ces dernières années un retour en force dans le domaine sur la

validation (ou revalidation) des métriques comme indicateurs de certaines caractéristiques de la qualité, d'une part, et sur le développement d'approches permettant (entre autres) d'identifier des « noyaux » pertinents de métriques pouvant être utilisés efficacement pour prédire la qualité des logiciels, d'autre part. En effet, un des problèmes rencontrés ces dernières années est relié au fait qu'il existe une multitude de métriques qui ont été définies (et implémentées) et qu'il est difficile, pour un gestionnaire ou un programmeur tout simplement, de déterminer, en fonction de ses objectifs d'évaluation, quel est le sous ensemble pertinent de métriques à utiliser. Sans compter le fait que les utiliser toutes engendre inévitablement une quantité d'information très importante qui contient certainement des redondances et qu'il est difficile de gérer et encore moins d'assimiler.

Dans un deuxième temps, nous présenterons notre modèle, les travaux sur lesquels il est basé (origine), ses idées maîtresses, sa construction, l'environnement au complet (qui a été développé) qui supporte son évaluation, et ses extensions possibles. Ces dernières seront formulées suite aux conclusions des premières évaluations empiriques conduites dans le présent projet de recherche. Cela justifiera l'orientation de nos principales et plus importantes expérimentations qui seront décrites dans la troisième partie.

Enfin, dans un troisième temps, nous regroupons nos différentes expérimentations en 3 grandes familles à savoir :

- L'analyse par composantes principales qui nous montrera jusqu'à quel point notre modèle peut se substituer a certaines métriques de bases (connues et largement utilisées dans la pratique), et nous renseignera sur le degré de corrélation qui existe entre notre modèle et les métriques existantes.

- La prédiction des changements confrontera notre modèle à certaines métriques reconnues dans la littérature et utilisées pour prédire les changements des logiciels (sur plusieurs générations), et ceci par différentes approches de mesure statistique. Nous verrons pour certains logiciels, la qualité de la prédiction des fautes.

- En fin, l'expérimentation sur la testabilité nous donnera les résultats des mesures de la testabilité selon certains modèles déjà utilisés dans la littérature. Nous énoncerons par la suite, et toujours dans le cadre de la maintenance, une manière d'utiliser notre modèle comme outil d'aide aux tests d'intégration, à la détection des cycles, et à l'analyse de l'impact des changements.

Nous terminerons ce mémoire par une conclusion globale dans laquelle, entre autres, nous poserons des questions restées ouvertes qu'il serait intéressant d'investiguer dans le cadre de travaux futurs.

CHAPITRE 2

ÉTAT DE L'ART :

UN BREF APERCU SUR CERTAINES MÉTRIQUES ORIENTÉES OBJET ET LEUR UTIILISATION

Dans ce chapitre, nous dressons un bref état de l'art de certains éléments relatifs à la métrologie des logiciels orientés objet. Le contenu, et les nombreuses recherches effectuées dans ce domaine, montrent l'intérêt de la communauté, et en particulier celui des gestionnaires, à disposer de moyens de mesure objectifs pour appréhender la qualité logicielle, surtout dans le contexte des applications industrielles complexes qui se développent de nos jours. Ce domaine aussi a suscité un grand dénouement relativement à l'évaluation de programmes construits en utilisant les paradigmes traditionnels. Il faut aussi dire que l'intérêt de l'évaluation de la qualité des programmes, avec leur complexité grandissante et les nombreux enjeux qui y sont rattachés, ne cesse de grandir.

Cette partie met aussi en relief les travaux les plus intéressants et prometteurs dans ce domaine. Elle présente également des études récentes ou en cours sur la validation des métriques et l'automatisation de l'évaluation de la qualité de la conception des logiciels. Nombreuses sont les propositions de métriques pour les systèmes orientés objet qui ont vu le jour depuis le début des années 90. L'une des premières propositions concrètes fut la suite de métriques OO proposée par Chidamber et Kemerer [Chidamber 94]. Cette proposition est en effet systématiquement citée dans les bibliographies de la grande majorité des travaux importants dans le domaine des métriques orientées-objet. Les travaux récents les citent également. Elle est également utilisée industriellement par des

agences comme la NASA [Rosenberg 98] et intégrée dans certains outils relatifs à la qualité (AOPmetrics : plugin d'Eclipse). Certains des éléments proposés à l'origine ont été repris et corrigés dans des versions plus récentes [Chidamber 98]. Cette suite de métriques a été largement, avec plusieurs autres propositions, expérimentée et discutée dans la littérature [Hendersen 96].

Ces métriques ont d'ailleurs fait l'objet d'analyses et de critiques directes, suite à leur évaluation, par de nombreux chercheurs tels que [Churcher 95], [Hitz 96], [Henderson 96] et indirectes [Kitchenham 95]. L'approche ascendante employée n'est pas elle-même exempte de réserves. Cette suite de métriques ne peut donc être considérée comme l'aboutissement des recherches dans ce domaine. Elle est certes une des premières propositions intéressantes dans ce domaine, mais constitue à notre avis un premier point de départ. C'est pourquoi d'autres travaux plus récents seront également présentés.

2.1 Les métriques proposées par Chidamber et Kemerer

Ces métriques visent, essentiellement, à mesurer la complexité de la conception des classes dans un système orienté objet. Afin d'évaluer la validité théorique de ces métriques, Chidamber et Kemerer les ont vérifiées (évaluation théorique) à l'aide de critères s'appuyant sur 9 propriétés définies par Weyuker. Les métriques proposées vérifient toutes ces règles à l'exception de :

- celle dénommée " interaction increase complexity ", aucune métrique ne vérifie cette propriété.
- celle dénommée "monotonicity", les métriques DIT et LCOM (présentées ultérieurement) ne vérifient pas cette propriété.

Nous présentons, dans ce qui suit, ces métriques telles que leurs auteurs les ont proposées, notamment en ce qui concerne les attributs prétendument mesurés.

2.1.1 WMPC - Weighted Methods Per Class

Définition : C'est le nombre de méthodes définies dans une classe.
Justification théorique : WMPC est un reflet de la complexité.
Attributs mesurés :

- Prédiction du temps et de l'effort nécessaires au développement et à la maintenance d'une classe. Plus une classe a de méthodes, plus elle demandera de travail.
- Impact sur la réutilisabilité, par le nombre de méthodes dont vont hériter les classes dérivées.
- Évaluation de la réutilisabilité. Un WMC élevé étant le signe d'une limitation de cet attribut.

Le nombre de méthodes et leur complexité permettent de prévoir le temps et l'effort requis pour développer et maintenir la classe. La complexité cyclomatique est liée au taux de fautes. Un WMPC élevé est synonyme d'un risque élevé de fautes dans la classe, mais aussi d'une compréhensibilité plus difficile [Basili 96].

2.1.2 DOIH - Depth Of InHeritance tree

Définition : C'est le nombre maximum de classes ancêtres de la classe pour atteindre la (une) racine.
Justification théorique : DIT est un reflet de la complexité via la portée des ancêtres.
Attributs mesurés :

- Évalue la complexité de la classe. Le comportement de la classe étant plus difficile à comprendre quand le nombre de méthodes héritées croît.

- Évalue la complexité globale. La conception est plus complexe puisqu'il y a plus de classes et de méthodes à développer.
- Évalue la réutilisabilité de la classe. Plus une classe est loin dans la hiérarchie, moins elle est générique. Elle peut renseigner un degré de spécialisation important qui fait que la classe s'éloigne de l'abstraction initiale.

Le mécanisme d'héritage introduit par la technologie objet permet de garantir une meilleure réutilisation, structuration et encapsulation du code. Cependant, l'héritage doit être utilisé de manière minimale et suffisante. Car, d'une part, l'absence d'héritage est souvent synonyme de manque de réutilisation de code et, d'autre part, des arbres d'héritage très profonds affectent la maintenabilité, la testabilité et lisibilité d'un code.

2.1.3 NOC - Number Of Children

Définition : C'est le nombre de classes immédiatement dérivées d'une classe.
Justification théorique : NOC est un reflet de l'impact potentiel d'une classe sur ses descendants.
Attributs mesurés :

- Évalue la réutilisabilité. Une classe ayant de nombreux enfants étant très générique.
- Évalue une mauvaise abstraction. Une classe ayant de nombreuses classes dérivées ayant une plus grande probabilité d'être improprement abstraites.
- Évalue l'influence sur le système et sur l'effort de test. Une classe ayant de nombreuses classes dérivées a un impact fort sur la hiérarchie de classes. Un effort de test particulier devra lui être appliqué.

2.1.4 CBO - Coupling Between Object classes

Définition : Nombre de couplages (degré de dépendances) entre une classe et toutes les autres classes du système (invocation de méthodes ou de variables).
Justification théorique : CBO est un reflet de l'interdépendance entre les constituants du système.
Attributs mesurés :

- Évalue la modularité et la réutilisabilité. Plus une classe est couplée aux autres, moins elle est modulaire et réutilisable.
- Évalue la modularité et l'effort de maintenance. Plus une classe est couplée aux autres, plus une modification de celle-ci risque d'affecter d'autres parties du système (et éventuellement inversement selon le sens des dépendances).
- Évalue la testabilité. Plus une classe est couplée aux autres, moins il sera aisé de vérifier toutes les interactions possibles.

Des études [Briand 99] ont montré qu'un couplage excessif entre les classes nuit à la modularité et constitue un obstacle à la réutilisation. Plus une classe est indépendante, plus il est facile de la réutiliser dans une autre application. Afin d'améliorer la modularité et de favoriser l'encapsulation, le couplage de classes interclasses doit être le plus limité possible. Plus le nombre de couples est élevé, plus les autres parties de la modélisation sont sensibles aux modifications et plus la maintenance est difficile. Par ailleurs, la mesure du couplage s'avère utile pour prévoir le niveau de complexité des tests pour les différentes parties d'une modélisation. Plus le couplage de classes inter-objets est important, plus les tests doivent être rigoureux.

2.1.5 RFC - Response For a Class

Définition : C'est le nombre de méthodes (de la classe et d'autres classes) potentiellement appelées par une classe en réponse à un message.

Justification théorique : RFC est un reflet du niveau de communication potentiel entre une classe et les autres.
Attributs mesurés :

- Évaluation de la testabilité. Plus une classe invoque des méthodes de diverses origines, plus elle est compliquée à comprendre.
- Évaluation de la complexité. Plus une classe invoque de méthodes, plus elle est complexe.
- Évaluation de l'effort de test. Plus une classe est complexe, plus l'effort de test sera important.

Une classe qui fournit un ensemble « réponse » plus grand est considérée comme plus complexe et comme nécessitant plus de test qu'une classe dont la complexité de la modélisation générale est plus faible.

2.1.6 LCOM - Lack of COhesion in Methods

Définition : C'est le nombre de méthodes prisent deux à deux (paires de méthodes) ne partageant pas des instances de variables de la classe. Moins le nombre de paires de méthodes partageants des instances de variables de la classe, moins la classe est cohésive. Si cette valeur est négative, LCOM est fixée (selon la définition des auteurs) égale à zéro.
Justification théorique : une classe est cohésive si ses méthodes agissent sur le même ensemble de données. Les méthodes sont donc reliées entre elles. Elle renseigne la qualité de la structure de la classe. Une faible cohésion indique éventuellement que la classe a été investie de responsabilités disparates.
Attributs mesurés :

- Évalue l'encapsulation. Une classe cohésive promouvant celle-ci.
- Évalue les défauts de conception de classes. Une classe peu cohésive devant sans doute être éclatée en plusieurs autres classes plus cohésives.

- Évalue la complexité. Une classe disparate augmente la probabilité d'erreur durant le développement.
- Renseigne la réutilisabilité : une classe non cohésive est faiblement réutilisable.

Une valeur forte indique un faible échange de donnée des méthodes de la classe d'où une perte d'unité structurelle de la classe, ce qui nuit grandement à la réutilisabilité de la classe. [Basili 96].

2.1.7 Critique des métriques proposées par Chidamber et Kemerer

Ces métriques, validées empiriquement par leurs auteurs, l'ont été également par plusieurs auteurs après leur publication. Une des premières études conduites dans ce sens a été réalisée par Basili et al. [Basili 96]. Cette étude menée sur 8 équipes d'étudiants de l'université du Maryland, visait à vérifier que les métriques proposées prédisaient la proportion d'une classe à générer des défauts. Elle conclut que 5 des 6 métriques proposées semblent des prédicateurs utiles (la métrique LCOM étant invalidée). Les auteurs préconisent une validation plus représentative de ces métriques dans le cadre de véritables projets industriels.

Par ailleurs, Churcher et al. [Churcher 95] expriment des réserves sur ces métriques. Tant d'un point de vue historique (travaux d'Halstead) que d'un point de vue scientifique. Ceci concerne notamment le défaut de méthodologie précise de mesure. Par exemple, suivant le mode de comptage employé, une classe C++ (simple) peut voir la métrique WMC varier de 12 à 37.

Hitz et al. [Hitz 96] quant à eux montrent les limites de la formulation des métriques CBO (qui ignore notamment la loi de Demeter) et LCOM. Cette dernière n'étant valide à leurs yeux qu'avec le mode de calcul proposé par W. Li et al [Li93]. Sinon, il faudrait expliquer des situations très paradoxales.

Notamment celle résultant de l'ajout d'une méthode sur des jeux déjà constitués de méthodes partageant des attributs de la classe. Il peut se produire alors :

- Soit une augmentation de la cohésion de la classe ;
- Soit une diminution de la cohésion de la classe ;
- Soit, ne pas changer la cohésion de la classe.

De plus, la validité théorique du crible des 9 propriétés de Weyuker, auxquelles ont été soumises ces métriques, a été remises en cause, notamment par B.Kitchenham et al. [Kitchenham 95]. Plusieurs interprétations des propriétés évaluées par les métriques proposées par les auteurs sont peu convaincantes. Par exemple, et contrairement à ce qu'ils affirment, rien n'indique qu'une classe à fort WMC serait mal aisée à réutiliser; et réciproquement.

L'utilisabilité de ces métriques en phase de conception, c'est-à-dire avant le codage (ce qui fait la force principale des métriques orientées objet), est inégale. En effet, la mesure des métriques CBO, RFC et LCOM implique que toutes les classes du système soient déjà implémentées. Ceci limite sévèrement leur emploi aux seules phases terminales du cycle de vie du logiciel (test, maintenance). Plusieurs particularités fondamentales de la conception des systèmes orientés objet ne sont pas ou peut capturées :

- Abstraction (classes abstraites par exemple) ;
- Encapsulation (parties privées d'une classe par exemple) ;
- Polymorphisme (méthodes virtuelles d'une classe) ;
- Service offert (partie publique d'une classe) ;
- Terminaison d'arborescence (classes non dérivables) ;
- Configurations spécifiques des classes proches et éloignées de la racine.

Enfin, Chidamber et Kemerer bien que promoteurs d'une approche initialement théorique, adoptent plutôt une démarche ascendante. Ainsi, ils définissent dans

un premier temps un certain nombre d'indicateurs, puis dans un second temps tentent d'évaluer ce qu'ils révèlent. C'est-à-dire, de quel attribut de quelle entité ils sont l'expression. Cette approche foncièrement empirique semble peu propice à faire émerger les métriques relatives aux différents attributs de la qualité d'une conception logicielle. En effet, au moins trois problèmes se posent ici :

- Comment reconnaître si une métrique est pertinente parmi la quasi-infinité de métriques qu'il est possible d'imaginer ? Selon quelles procédures, selon quels critères juger du support sémantique d'une métrique ?

- Dans l'éventualité où la question précédente serait résolue, disposant donc de métriques, comment "organiser" celles-ci de manière à disposer d'une évaluation qualitative de plus haut niveau d'abstraction ?

- Comment conduire les recherches dans la voie de métriques orientées vers un attribut qualitatif particulier ?

2.2 Les métriques proposées par Li et Henry

Cette proposition reprend cinq des six métriques de Chidamber et Kemerer, en y ajoutant cinq nouvelles. Parmi ces dernières, deux sont centrées sur la mesure du couplage entre objets, les trois autres évaluant leur taille.

2.2.1 Métriques reprises des propositions de Chidamber et Kemerer

Il s'agit des métriques DIT, NOC, RFC, LCOM et WMC. La métrique CBO n'étant pas retenue dans cette suite puisque d'autres sont proposées pour évaluer le couplage. Le couplage par héritage est quant à lui estimé correctement, appréhendé par les indicateurs DIT et NOC. Le mode de calcul de la métrique LCOM étant modifié pour devenir : le nombre de jeux disjoints de méthodes

locales à la classe dont au moins une instance de variable de classe est partagée entre elles.

2.2.2 MPC - Message Passing Coupling

Définition : nombre de messages envoyés par une classe en direction des autres classes du système (nombre de méthodes invoquées).
Justification théorique : le nombre de messages envoyés par une classe peut indiquer combien l'implémentation de ses méthodes dépend des autres classes.
Attributs mesurés : Évalue le couplage sortant d'une classe.

2.2.3 DAC - Data Abstraction Coupling

Définition : la définition de cette métrique est ambiguë. C'est pourquoi les deux interprétations possibles sont fournies :

- nombre de types de données abstraites définies dans une classe (classes dont la définition est incluse dans la définition d'une autre classe), ou
- attribut d'une classe qui est une autre classe.

La deuxième définition semble la plus cohérente, mais le texte des auteurs accréditerait plutôt la première.
Justification théorique : le nombre de classes ainsi 1) définies ou 2) utilisées, indique la dépendance d'une classe à la définition d'autres classes.
Attributs mesurés : Évalue le couplage "interne " d'une classe avec d'autres classes.

2.2.4 NOM - Number Of local Methods

Définition : nombre de méthodes localement définies dans une classe (hors méthodes héritées).

Justification théorique : ce nombre indique l'incrément de l'interface apporté par cette classe.
Attributs mesurés :

- Évalue les propriétés opérationnelles d'une classe (interface).
- Évalue la complexité de la classe : plus NOM est élevé, plus une classe est complexe.

Si une classe a beaucoup d'opérations, elle est difficilement réutilisable et perd souvent de sa cohésion. Il peut être judicieux de la diviser en sous-classes.

2.2.5 SIZE1

Définition : nombre d'instructions dans l'implémentation d'une classe.
Justification théorique : ce nombre est directement dérivé de la métrique traditionnelle LOC (Lines Of Code).
Attributs mesurés : Évalue la complexité d'une classe.

2.2.6 SIZE2

Définition : cumul du nombre d'attributs et du nombre de méthodes locales (NOM) d'une classe.
Justification théorique : ce nombre est indirectement dérivé de la métrique traditionnelle LOC (Lines Of Code).
Attributs mesurés : Évalue la complexité d'une classe.

2.2.7 Critique des métriques proposées par Li et Henry

Cette suite incorporant la plupart des métriques définies par Chidamber et Kemerer, les critiques relatives à celles-ci s'appliquent également. La validation de ces métriques par les auteurs met en évidence une forte corrélation de celles-ci prisent toutes ensemble avec l'effort de maintenance qu'il a fallu déployer

pendant 3 ans sur 2 applications industrielles écrites en Classic-ADA. Cette étude met en évidence que les métriques SIZE1 et SIZE2 sont d'importants indicateurs. Mais également qu'un modèle réduit sans ces 2 métriques pouvait également servir de prédicateur de l'effort de maintenance, bien qu'avec une moins grande fiabilité. La formulation de la métrique DAC est ambiguë.

S'il semble avéré que ces métriques (incluant celles de Chidamber et Kemerer) soient efficaces pour prévoir l'effort de test et de maintenance, leur utilisabilité en phase de conception demeure problématique. La raison essentielle de cet état de fait étant leur indisponibilité en dehors de tout codage : c'est le cas des métriques RFC, LCOM, MPC, SIZE1 et suivant le mode de calcul de WMC.

2.3 Les métriques proposées par Abreu, Goulão et Esteves (MOOD)

Le projet MOOD (Metrics for Object Oriented Design) [Abreu 95] consiste en une proposition de 6 métriques dont les principales caractéristiques sont :

- Une forte corrélation avec les concepts objet ;
- Une évaluation d'un système dans sa globalité ;
- Une évaluation possible en dehors de toute implémentation ;
- Une expression en pourcentage, éliminant les questions de signification quant à la valeur d'une métrique.

Ces métriques reposent sur le principe suivant :

Métrique = Nombre d'occurrences dans le système/ Nombre maximal d'occurrences dans le système.

Elle s'appuie sur l'hypothèse que la mesure de fréquence d'emploi de certains facteurs de construction orientée objet reflète la qualité de la conception. Ces métriques ont été jaugées sur des réalisations commerciales prises comme étalons d'une bonne conception orientée objet. Il en résulte des

recommandations quant aux fourchettes dans lesquelles doivent se trouver chacune de ces métriques.

2.2.8 MHF - Method Hiding Factor

Définition et Justification théorique : pourcentage de méthodes cachées.
Attributs mesurés : Évalue l'encapsulation.
Fourchette préconisée : [10%, 30%]

2.2.9 AHF - Attribute Hiding Factor

Définition et Justification théorique : pourcentage d'attributs cachés.
Attributs mesurés : Évalue l'encapsulation.
Fourchette préconisée : [70%, 100%]

2.2.10 MIF - Method Inheritance Factor

Définition et Justification théorique : pourcentage de méthodes héritées.
Attributs mesurés :

- Évalue l'abstraction.
- Évalue la fonctionnalité.

Fourchette préconisée : [65%, 80%]

2.2.11 AIF - Attribute Inheritance Factor

Définition et Justification théorique : pourcentage d'attributs hérités.
Attributs mesurés : Évalue l'abstraction.
Fourchette préconisée : [50%, 60%]

2.2.12 PF - Polymorphic Factor

Définition et Justification théorique : pourcentage de méthodes polymorphes par rapport au nombre total de méthodes potentiellement polymorphes.
Attributs mesurés : Évalue la flexibilité.
Fourchette préconisée : [3,5%, 10%]

2.2.13 COF - Coupling Factor

Définition et Justification théorique : pourcentage de classes couplées aux autres classes autrement que par l'héritage.
Attributs mesurés : Évalue l'interdépendance.
Fourchette préconisée : [4%, 20%]

2.2.14 Critique des métriques MOOD

Contrairement aux métriques précédentes, celles-ci présentent des points positifs notables :

- Elles sont basées sur les fondements du paradigme orienté objet ;
- Elles sont théoriquement indépendantes du langage de mise en œuvre ;
- Elles s'expriment en pourcentage donc dans une unité largement connue ;
- Elles présentent un caractère opérationnel puisque des plages de variations sont préconisées ;
- Elles peuvent être calculées en dehors de toute implémentation, donc dès les premières phases du cycle de vie du logiciel.

Certaines de ces métriques peuvent être utilisées à plusieurs niveaux de granularité : système, hiérarchie, classe. Ces métriques sont utilisables et interprétables (grâce à une expression en pourcentage et à la fourniture de fourchettes) dès les premiers temps d'un développement itératif. Une validation sur une plus grande échelle est nécessaire pour s'assurer de leur efficacité. En

effet, ces métriques n'ont été testées que dans le cadre de [Abreu 96], celui d'un groupe d'étudiants. Cette validation a montré le caractère prédictif sur la fiabilité et la maintenabilité des métriques MHF, AIF, MIF, POF et COF. Elle montre également de manière surprenante que AHF serait peu significative. Comme pour les métriques de Chidamber et Kemerer, l'absence d'un cadre qualitatif de haut niveau est une limitation.

Un autre groupe de métriques a été également proposé, à cette période, par Badri et al. [Badri 95, Badri 96]. Ces métriques présentaient l'avantage d'être applicables dès les premières phases du processus de développement. Elles ont été évaluées théoriquement et n'ont pas fait l'objet d'expérimentation de grande envergure. La métrique de cohésion proposée dans [Badri 95] a été cependant reprise et largement expérimentée. Les études empiriques effectuées [Badri 03, Badri 04] ont largement démontré que l'approche de cohésion OO proposée par Badri et al. permettait de capturer plus de paires de méthodes reliées que les métriques de cohésion proposées dans la littérature. Ces études ont permis d'étendre les critères de cohésion OO. Une étude récente a permis de démontrer la forte corrélation qui existe entre la cohésion, telle que définie par Badri et al., et le couplage [Badri 08].

2.3 Prédiction des changements

N. Tsantalis et al. dans leurs travaux [Tsantalis 05] utilisent les probabilités pour prédire les changements dans les systèmes orientés objet. Leur modèle probabiliste repose sur la formule de Bayes. Leur analyse leur a permis d'identifier 2 axes de propagation pour le changement: un axe d'héritage, soit par l'implémentation d'interface soit par l'héritage d'une classe; un axe de référence principalement lié au couplage. Le modèle prédit la probabilité qu'a une classe de subir des changements. Il donne une bonne concordance avec la réalité dans le cas ou le taux de propagation est raisonnable. L'expérience qu'ils

ont menée sur des logiciels open source a permis de montrer la supériorité de leur approche par rapport aux métriques traditionnelles dans la prédiction des changements. Les résultats fournis par leur métrique (modèle) est une probabilité comprise en 0 et 1, ce qui facilite leur interprétation. Cependant, le modèle n'est applicable qu'aux classes et non aux méthodes, ce qui pourrait être judicieux dans certains cas. D'un autre coté, le modèle est purement prédictif, et ne capture aucun attribut de qualité, ou du moins, aucune expérimentation n'a été menée dans ce sens.

2.4 Recoupement des métriques

Un des problèmes des métriques, discutés par plusieurs chercheurs, est lié au recoupement qui existe entre plusieurs d'entre elles. Les métriques ne sont pas linéairement indépendantes. Dans ce cadre, Aggarwal et al [Aggarwal 06] ont tenté d'étudier les métriques objets afin d'isoler un groupe de métriques plus restreint capable de remplacer l'ensemble des métriques existantes. En appliquant une analyse par composante sur le jeu de mesures obtenues sur trois logiciels différents, ils on réussi à isoler un certains nombre de métriques qui capturent la majorité de l'information contenue dans leur ensemble de mesures. Ils ont, par ailleurs, réussi à donner un sens aux 3 premières composantes de l'ACP, en remarquant que la première colonne identifiait les métriques de taille, la deuxième était reliée à la cohésion, et la quatrième à l'héritage ; les autres composantes étant plus difficiles à interpréter. Malheureusement, dans leur étude, l'analyse n'a été effectuée que sur des logiciels de petite taille, d'aide à la décision. Aurait-on obtenu le même résultat si on était parti sur un logiciel de graphisme, ou un parseur par exemple ?

Le modèle que nous présentons dans ce mémoire regroupe plusieurs métriques de base, de manière directe ou indirecte. Il capture des attributs de qualité de haut niveau, et constitue un outil de support pour la phase de maintenance des logiciels.

CHAPITRE 3

LE MODELE : VERS UN INDICATEUR DE QUALITÉ UNIFIANT PLUSIEURS DIMENSIONS

Le modèle que nous proposons, est un affinement, pour le paradigme objet, du modèle original, comme mentionné précédemment, proposé par Badri et al. [Badri 95] pour supporter, entre autres, les tests d'intégration dans les systèmes orientés objet. A l'origine, ce modèle a également été utilisé pour le paradigme de programmation impérative. Il est basé sur une analyse statique du code et une affectation de probabilité aux appels de méthodes en fonction du chemin suivi par le flux d'exécution. L'instance du modèle que nous avons implémentée permet l'analyse du code Java uniquement, mais peut s'étendre facilement aux autres langages orientés objet. Une valeur comprise entre 0 et 1, que nous appellerons Indicateur de Qualité (Qi), est affectée à chaque classe du système à la fin du calcul. Notons que l'interdépendance des Qi des méthodes engendre un système d'équations que nous devrons résoudre automatiquement par la méthode numérique des approximations successives.

3.1 Construction du modèle

La construction du modèle [figure 1] repose, en partie, sur le graphe de contrôle réduit aux appels, et se fait en trois grandes étapes:

- La première étape consiste en l'analyse statique du code source et à la génération du graphe de contrôle réduit aux appels des différentes méthodes de classes;

- La deuxième étape nous permet de transformer le graphe d'appels de chaque méthode en une ligne d'équation permettant de calculer l'indicateur de qualité (une sorte d'indicateur de confiance) de la méthode en fonction des indicateurs de qualité de celles qu'elle appelle et de la probabilité qu'elle les appelle; Les valeurs obtenues sont surtout considérées de façon relative. Elles permettent de déterminer, entre autres, les éléments les plus critiques d'une conception.

- Enfin, dans une troisième étape, nous résolvons ce système pour obtenir les valeurs des Qi de l'ensemble des méthodes du système.

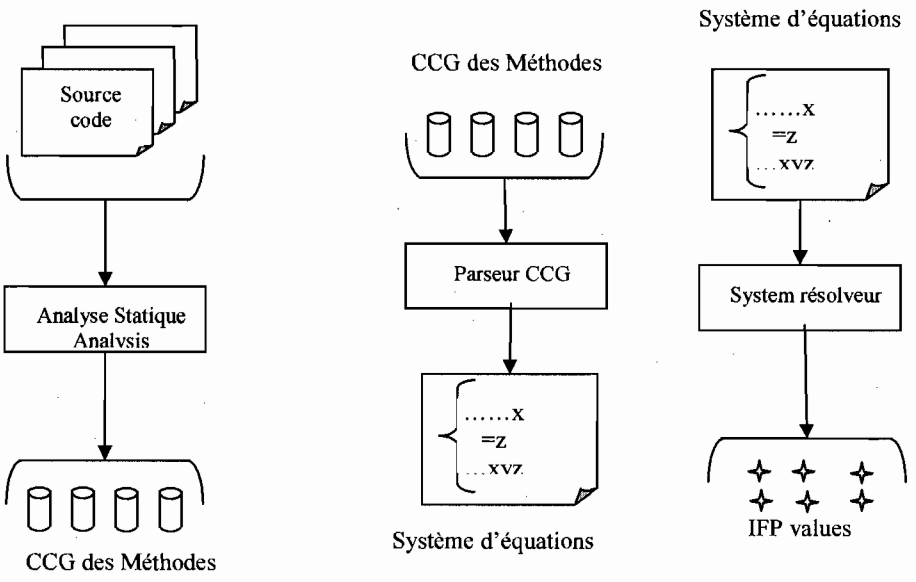


Figure 1 : Schématisation du principe du modèle

3.1.1 Analyse du code source et construction du graphe d'appels réduit

La première étape de la construction du modèle est une analyse complète de toutes les méthodes de toutes les classes du code source. Grâce à cette analyse,

nous dressons un graphe de contrôle réduit aux appels qui synthétise les chemins du flux de contrôle dans chaque méthode.

3.1.1.1 Graphes de contrôle réduits aux appels

Pour définir un graphe de contrôle réduit aux appels, nous allons définir le graphe de flux de contrôle dont il découle.

Définition 1 : Un graphe de flux de contrôle (CFG) est un graphe orienté. Les nœuds de ce graphe représentent soit des structures de contrôle (if-then-else, while, case, etc.), soit une instruction ou un bloc séquentiel d'instructions. Un bloc séquentiel d'instructions est une séquence d'instructions telle que si nous exécutons la première instruction, nous sommes certains d'exécuter les autres, et toujours dans le même sens. Un arc orienté lie un nœud N_i à un nœud N_j s'il est possible d'exécuter l'instruction correspondante à N_j immédiatement après celle associée au nœud N_i. Les arcs du graphe indiquent le transfert de contrôle d'un nœud à l'autre.

Définition 2 : Un graphe de contrôle réduit aux appels (CCG) est un graphe de flux de contrôle (CFG) dans lequel sont éliminés les nœuds représentant les instructions qui ne conduisent pas à des appels.

Considérons l'exemple de la méthode M donnée par la figure 2.1. Les S_i représentent des séquences d'instructions ne contenant pas des appels de méthodes. La synthèse de la méthode M réduite aux appels est donnée par la figure 2.2. Le graphe d'appels correspondant est donné par la figure 2.3. La figure 2.4 donne le graphe de contrôle réduit aux appels correspondant.

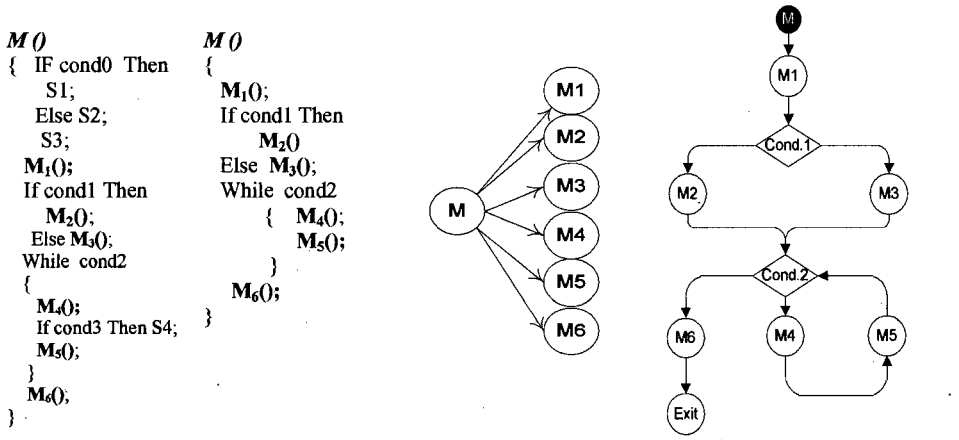


Figure 2: Portion d'une méthode et graphe d'appels et graphes de contrôle réduit aux appels correspondants

3.1.1.2 Polymorphisme et graphe de contrôle

Le polymorphisme qui est un concept important dans la technologie objet, est aussi le plus difficile à saisir par une simple analyse statique. En effet, les liaisons dynamiques font que l'appel effectif vers une des méthodes virtuelles ne se fait que lors de l'exécution du code. Les messages polymorphes figurant dans les graphes de contrôle des méthodes sont marqués. Un nœud correspondant à un appel polymorphe est représenté par un polygone de n+1 cotés, n étant le nombre de méthodes pouvant éventuellement répondre à cet appel. Les sommets de ces polygones sont liés par des arcs orientés, aux méthodes virtuelles de la liste des méthodes pouvant potentiellement répondre à l'appel. Cette dernière est déterminée lors de l'analyse statique du code source.

Toujours par rapport à l'exemple précédent, on considère que l'appel m2 est polymorphe et que la méthode numérotée m21 peut aussi répondre à la

place de m2. L'extension du graphe d'appel pour tenir compte du polymorphisme donne la figure 3.

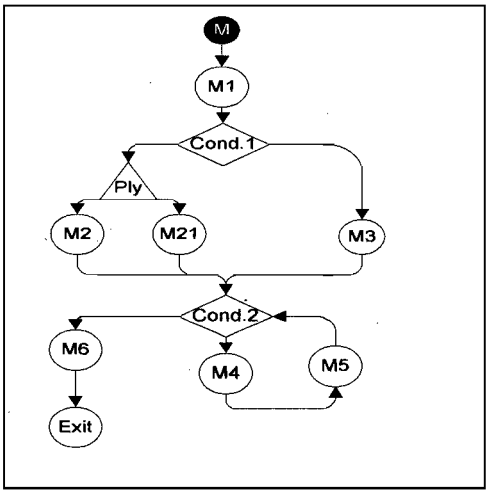


Figure 3:Graphe contrôle réduit aux appels da la même méthode intégrant les appels polymorphiques

Représenté de cette manière, on pourra affecter une probabilité d'appel aux appels polymorphes.

3.1.2 Analyse du graphe d'appels et construction du système d'équations

Dans un graphe de contrôle réduit aux appels, on appellera chemin l'ensemble connexe d'arcs qui relient le début à la fin du graphe.

Le CCG d'une méthode peut être vu comme un ensemble de chemins que peut emprunter le flux de contrôle pour traverser la méthode. Comme le flux ne peut passer par tous ces chemins en même temps, nous affectons une probabilité de passage pour chaque chemin particulier du graphe.

3.1.2.1 Affectation des probabilités d'appels

Pour une méthode donnée, le flux de contrôle suit un chemin parmi tous ceux qui sont représentés dans le CCG. L'emprunt de ce chemin dépend des

conditions dans les structures de contrôle. Pour capturer cette caractéristique probabiliste du flux, nous affectons une probabilité à chaque chemin défini par le CCG comme suit :

$$\text{Pour un chemin } C_k, P(C_k) = \prod_{i=0}^{n_k} P(A_i),$$

où les A_i sont les arcs qui composent le chemin C_k..

En supposant (pour simplifier l'analyse et les calculs) que les conditions dans les boucles sont indépendantes les unes des autres, les $P(A_i)$ deviennent les probabilités qu'un arc soit emprunté à la sortie d'une structure de contrôle. Les $P(C_k)$ se réduisent alors au produit des probabilités des états des conditions dans les structures de contrôle.

Nous avons affecté des probabilités à ces structures de contrôle de la manière suivante (voir Tableau 1).

Nœuds	Affectation de probabilité
(If, else)	0.5 pour l'arc de sortie condition = vrai 0.5 pour l'arc de sortie condition=faux
while	0.75 pour l'arc de sortie condition = vrai 0.25 pour l'arc de sortie condition=faux
(Do, while)	1 pour l'arc :(les instructions internes sont exécutées au moins une fois)
(Switch,case)	1/n pour les chaque arc des n cases.
(?, :)	0.5 pour l'arc de sortie condition= vraie pour l'arc de sortie condition=faux
for	1 pour l'arc
(try, catch)	0.75 pour l'arc du bloc try 0.25 pour l'arc du bloc catch
Polymorphisme	1/n pour chacun des n appels éventuels.

Tableau 1:Règle d'affectation des probabilités aux structures de contrôle.

Considérons le CCG de la figure2, on constate qu'il y a 6 chemins pour le flux de contrôle qui sont :

$C_1 : M_1, M_2, M_6$
$C_2 : M_1, M_2, M_4, M_5, M_6$
$C_3 : M_1, M_{21}, M_6$
$C_4 : M_1, M_{21}, M_4, M_5, M_6$
$C_5 : M_1, M_3, M_6$
$C_6 : M_1, M_3, M_4, M_5, M_6$

La probabilité associée au chemin C1 en tenant compte des structures de contrôle est donnée par :

$$P(C_1)=1*0.5*0.5*1*0.5*1 =0.125$$

Le produit des probabilités vient du fait que les conditions dans les structures doivent être dans un état (vrai/faux) précis pour que le flux emprunte ce chemin.

3.1.2.2 Indicateur de Qualité / Indicateurs de fiabilité prévisionnelle.

Pour une méthode M_i, nous définissons son Qi (indicateur de qualité) appelé également IFP (indicateur de fiabilité prévisionnelle – nom utilisé dans les travaux antérieurs de Badri et al.) comme une estimation de la probabilité que le flux de contrôle traverse la méthode sans « défaillance ». Les IFP ont été développés et utilisés, comme mentionné ultérieurement, dans les travaux de Badri et al. pour supporter, entre autres, le processus de test des systèmes OO. Nous pensons, cependant, que leur portée dépasse le processus de test et qu'ils peuvent (de part la formulation du modèle de base) être utilisés également pour prédire certaines dimensions de la qualité des systèmes orientés objet, et remplacer par la même plusieurs des métriques OO existantes. De cette idée découle le nom d'indicateur de qualité (Qi) plus large. Le Qi d'une méthode (ou indicateur de fiabilité prévisionnelle de la méthode) est fonction des Qi des méthodes qu'elle appelle, de la probabilité que ces dernières soient appelées et

d'une constante intrinsèque à la méthode M_i appelée indicateur de fiabilité intrinsèque (ou Qi intrinsèque) de la méthode.

$$IFPM_i = IFM_i^* * \sum_{j=1}^{n_i} \left(P\left(C_j^i\right) * \prod_{k \in \sigma} IFPM_k \right) \quad (1)$$

$IFPM_i$: Indicateur de fiabilité prévisionnelle de la méthode M_i

IFM_i^* : Indicateur de fiabilité intrinsèque de la méthode M_i

$P(C_j^i)$: Probabilité d'emprunt du chemin C_j^i de la méthode M_i

$IFPM_k$: IFP des méthodes se trouvant sur le chemin C_j^i

n_i : Nombre de chemins définis par le CCG de M_i

$Card(\sigma)=m_j$: Nombre de méthodes se trouvant sur le chemin C_j^i.

3.1.2.3 Indicateur de qualité intrinsèque (Indicateur de Fiabilité prévisionnelle intrinsèque)

L'indicateur de fiabilité intrinsèque (ou indicateur de qualité intrinsèque) d'une méthode regroupe d'autres paramètres caractérisant la fiabilité prévisionnelle (qualité prévisionnelle intrinsèque) qu'aurait cette méthode si toutes les méthodes qu'elle appelait avaient une fiabilité parfaite. Parmi les éléments caractéristiques de l'IFP intrinsèque (Qi intrinsèque) figurent :

- Sa complexité cyclomatique,
- L'effort de test unitaire (taux de couverture de test)
- L'effort d'assurance et de contrôle qualité appliqué.
-

L'indicateur de fiabilité intrinsèque se définit comme suit pour une méthode M_i :

$$IFM_i^* = \left(1 - \frac{tf_i}{tf_{\max}}\right) \quad (2)$$

Où tf_i est le taux de fautes résiduel de la méthode M_i $tf_i = VG \cdot \left(1 - tc_i\right)$;

Avec VG : La complexité cyclomatique de la méthode M_i,

tc_i : Taux de couverture de test unitaire de la méthode M_i, $tc_i \in [0,1]$

$$tf_{\max} = \max_{1 \le i \le N}(tf_i)$$

De la formule (1) et (2), on peut tirer deux remarques :

- $IFM_i^* \in [0,1]$.

- La formule (**1**) est implicite, cela traduit le fait que le calcul de l'IFP d'une méthode M_i nécessite la connaissance de la valeur des IFP des méthodes qu'elle appelle.

En posant simultanément ses équations pour chaque méthode, on obtient le système suivant :

$$\begin{cases} IFPM_i = IFM_i^* * \sum_{j=1}^{n_i} \left(P(C_j^i) * \prod_{k \in \sigma} IFPM_k \right) \\ i = 1....N \end{cases} \quad (3)$$

La taille de ce système est égale au nombre de méthodes que compte le logiciel. Il est non linéaire et est constitué de polynômes à plusieurs variables. L'existence et l'unicité de la solution sur [0,1] nous seront garanties par les transformations effectuées dans le chapitre 3.1.3.

3.1.3 Résolution du système d'équations

Le système d'équations obtenu précédemment (3) est non linéaire. Nous utiliserons une méthode itérative pour le résoudre, il s'agit de la méthode des approximations successives. A l'aide des transformations suivantes, on peut revenir à un problème du point fixe en définissant la fonction F :

$\mathbb{R}^N \rightarrow \mathbb{R}^N$

$$x \rightarrow F(x) = (f_1(x), f_2(x)....f_N(x))$$

$$avec \quad f_i(x) = IFM_i^* * \sum_{j=1}^{n_i} \left(P(C_j^i) * \prod_{k \in \sigma} x_k \right)$$

Résoudre le système d'équation de la (3) revient à trouver x tel que F(x) = x. L'existence et l'unicité de la solution découle du fait que F est contractante de rapport $K = \max_{1<=i<=N} IFM_i^*; K \leq 1$. En posant $x^{(n+1)} = F\left(x^{(n)}\right)$ on obtient que la suite $\left(x^{(n)}\right)_{n\geq 0}$ converge vers la solution pour n'importe quelle valeur initiale $x^{(0)}$ de la suite, en particulier pour $x^{(0)} = \left(1/2, 1/2, 1/2, 1/2 \ldots\ldots 1/2\right)$: Il s'agit de la méthode des approximations successives. La contraction de F garantit entre autre que toutes les composantes de la solution sont sur [0,1], autrement dit, les IFP sont des valeurs de l'intervalle [0,1]. La méthode permet d'obtenir rapidement une bonne approximation de la solution du problème.

Nous calculons les N itérations à effectuer pour obtenir une erreur de l'ordre de 10^{-5} grâce à la majoration de l'erreur donnée par :

$$e_n \leq K^n \left\| 1 - x^{(0)} \right\| \leq K^n \leq 10^{-5}$$

d'où, il suffit de prendre $N \geq -5\dfrac{\ln(10)}{\ln(K)}$

Notons que K est la constante de Lipschitz pour la fonction F définie plus haut.

On définira l'indicateur de qualité d'une classe Qi (indicateur de fiabilité prévisionnelle) comme étant le produit des Qi (IFP) de ses méthodes publiques. Cette formule nous vient de l'ingénierie de fiabilité. Le calcul des Qi ne tient pas compte du flux de données, de ce fait nous mesurerons toujours le rapport entre le flux de contrôle et le flux de données dans un système afin de s'assurer que le flux de contrôle est assez significatif pour pouvoir confronter les résultats de notre modèle avec la réalité lors des expérimentations, autrement dit, on vérifiera toujours que le principe de l'encapsulation est bien respecté.

3.2 Environnement et outils

Le calcul des indicateurs de qualité des classes est entièrement automatisé par un plug-in Éclipse que nous avons développé. Pour cela, notre outil utilise l'environnement de développement Éclipse ainsi que le logiciel de calcul scientifique Scilab.

3.2.1 Eclipse

C'est un environnement de développement intégré (IDE) dont le but est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques. I.B.M. est à l'origine du développement d'Eclipse qui est d'ailleurs toujours le cœur de son outil Websphere Studio Workbench (WSW), lui même à la base de la famille des derniers outils de développement en Java d'I.B.M. Tout le code d'Eclipse a été donné à la communauté par I.B.M afin de poursuivre son développement. Eclipse utilise énormément le concept de modules nommés "plug-in" dans son architecture. D'ailleurs, hormis le noyau de la plate-forme nommé "Runtime", tout le reste de la plate-forme est développé sous la forme de plug-ins. Ce concept permet de fournir un mécanisme pour l'extension de la plate-forme et ainsi fournir la possibilité à des tiers de développer des fonctionnalités qui ne sont pas fournies en standard par Eclipse. Les principaux modules fournis en standard avec Eclipse concernent Java mais d'autres plug-in existent pour d'autres langages notamment C++, Cobol, AspectJ, mais aussi pour d'autres aspects du développement (base de données, conception avec UML, ...). Ils sont tous développés en Java soit par le projet Eclipse soit par des tiers commerciaux ou en open source. Les modules agissent sur des fichiers qui sont inclus dans l'espace de travail (Workspace). L'espace de travail regroupe les projets qui contiennent une arborescence de fichiers. Bien que développé en Java, les performances à l'exécution d'Eclipse sont très bonnes, car il n'utilise pas Swing

pour l'interface homme-machine mais un toolkit particulier nommé SWT associé à la bibliothèque JFace. SWT (Standard Widget Toolkit) est développé en Java par IBM en utilisant au maximum les composants natifs fournis par le système d'exploitation sous jacent. JFace utilise SWT et propose une API pour faciliter le développement d'interfaces graphiques. Eclipse ne peut donc fonctionner que sur les plates-formes pour lesquelles SWT a été porté. Ainsi, Eclipse 1.0 fonctionne uniquement sur les plates-formes Windows 98/NT/2000/XP et Linux. SWT et JFace sont utilisés par Eclipse pour développer le plan de travail (Workbench) qui organise la structure de la plate-forme et les interactions entre les outils et l'utilisateur. Cette structure repose sur trois concepts : la perspective, la vue et l'éditeur. La perspective regroupe des vues et des éditeurs pour offrir une vision particulière des développements.

Le « plug-in » JDT supportant l'éditeur Java est basé sur les arbres de syntaxe abstraits (AST). En étendant ce plug-in et en utilisant les AST nous parvenons à analyser statiquement des systèmes et à générer un CCG. Nous avons par ailleurs ajouté un arbre supplémentaire pour tenir compte des appels polymorphiques.

3.2.2 Scilab

La résolution du système d'équations non linéaires de grande taille, généré automatiquement par notre outil après l'analyse du CCG se fait avec Scilab. Il s'agit d'un logiciel de calcul numérique scientifique qui fournit un puissant environnement de développement pour les applications scientifiques et l'ingénierie. Scilab est un logiciel distribué librement avec les sources sur Internet depuis 1994. Il est actuellement utilisé dans les entreprises, la recherche et l'enseignement dans le monde entier. Il est aujourd'hui développé par le Consortium Scilab, créé en 2003, lequel regroupe à ce jour 25 membres. Enfin, Scilab contient des centaines de fonctions mathématiques avec la possibilité de

rajouter interactivement des programmes écrits dans divers langages (FORTRAN, C, C++, JAVA ...). Il possède des structures de données sophistiquées (incluant les listes, les polynômes, les fractions rationnelles, les systèmes linéaires, ...), un interpréteur et un langage de programmation de haut niveau. C'est en utilisant son interface Java mise en place depuis sa version 4.0, qu'on communique de manière automatique avec ce logiciel à partir de notre plug-in.

CHAPITRE 4

L'OUTIL

4.1 Description générale de l'outil

Nous présenterons, dans ce chapitre, une vue générale des spécifications et du fonctionnement de l'outil. Nous détaillerons les contraintes d'implémentation et de performance, les cas d'utilisation ainsi que les modèles du domaine relatifs à notre outil (processus de conception). Nous finirons par dresser le diagramme d'état du plug-in vu comme entité objet.

4.2 Aperçu et exigences de l'outil

L'outil que nous avons implémenté est un plug-in pour l'environnement logiciel Eclipse (>3.1), écrit entièrement en Java (version >1.5). Il nécessite un minimum de puissance de calcul équivalent à celui d'un P4. De plus, 350Mo allouée à la JVM (« HeapSpace ») sont nécessaires pour l'analyse d'un logiciel d'environs 400 classes sous forme de projet fonctionnel pour Eclipse. Dans ces conditions, l'analyse se fera entre 2 et 5 secondes, selon les performances du processeur. La présence du logiciel Scilab (version 1.4) est nécessaire pour traitement des équations générées.

4.3 Vue d'ensemble des fonctions de l'outil

Afin d'automatiser le calcul des Qi ainsi que les applications du modèle développé plus haut, notre outil permet à son utilisateur de choisir des actions par l'intermédiaire d'un menu (GUI) interactif. Nous donnons dans ce qui suit la liste des actions permises par notre outil pour calculer et utiliser les Qi, classées par ordre d'importance.

- Analyser un projet Eclipse en vue de générer les CCG
- Modifier les taux de couverture de test des méthodes
- Calcul des Qi pour tous les composants : (Méthode et classes)
- Calcul de la criticité des composants (Méthodes et classes)
- Calcul de l'impact de changement d'une ou plusieurs méthodes sur le reste du système.
- Réinitialisation du parseur

4.4 Description des acteurs (utilisateurs)

Utilisateur : Il est le seul acteur humain pour cet outil. Sont objectif est d'obtenir des résultats sur la criticité des composants logiciels. Pour cela, il a les fonctionnalités définies plus haut pour arriver à ses fins.

Scilab : Ce logiciel externe à notre plug-in à pour mission de résoudre le système d'équations généré par l'outil selon le fichier script que celui-ci lui fournit. Sa version 4.0 est nécessaire pour assurer sa communication en Java avec notre outil.

Eclipse : Il s'agit de la plateforme de développement dans laquelle se greffe notre outil. Il assure une présentation des projets selon un format donné, compréhensible par notre outil, et nous fournit l'ensemble des bibliothèques nécessaires pour analyser le code source des projets.

4.5 Description détaillée

Ce chapitre décrit les exigences fonctionnelles détaillées de notre outil, regroupées par catégories de fonctions.

4.5.1 Spécification des cas d'utilisation

4.5.1.1 Le modèle des cas d'utilisation

Le modèle des cas d'utilisation, donné ci-dessous, montre l'interaction de l'utilisateur (Acteur principal) avec les autres environnements et outils (acteur secondaire) à travers les différents cas d'utilisation. Nous présentons dans ce qui suit les principaux cas d'utilisation (fonctionnalités importantes offertes par l'outil).

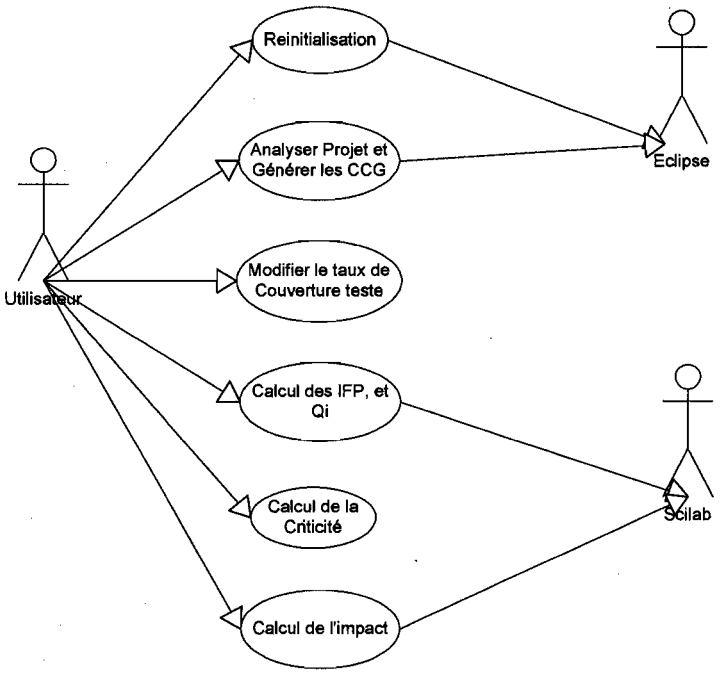


Figure 4: Le modèle des cas d'utilisations

Titre du cas d'utilisation:

Analyse d'un projet Eclipse et génération des CCG

Description sommaire:

L'utilisateur veut générer les graphes de contrôle réduits aux appels de toutes les méthodes définies dans un projet Eclipse écrit en Java.

Règles d'initiation :

- L'utilisateur doit avoir la plate forme Eclipse et le logiciel Scilab sur son PC
- Le client doit avoir le plug-in d'installé sur la liste des plug-ins d'Eclipse
- L'utilisateur doit avoir un projet à analyser sous le format standard d'Eclipse.
- Le projet à analyser doit être complet et fonctionnel

Description du processus :

1. L'utilisateur lance Eclipse
2. L'utilisateur ouvre le projet constituant le logiciel à analyser.
3. L'utilisateur ouvre le plug-in, et clique sur «Générer CCG »
4. Le plug-in retourne un message pour signaler la fin de l'analyse et la création de tous les graphes de contrôle.
5. Le plug-in affiche les unités de compilation ainsi que les CCG qu'elles contiennent pour chaque méthode sous forme d'arbre.

Règles de terminaison :

- Le graphe de contrôle généré est gardé en mémoire vive et est associé au projet.

Extensions :

A* - Une erreur est détectée lors de l'analyse :

Le système affiche un message d'erreur.

Le système journalise l'erreur et son origine.

Le système réinitialise toutes ses variables.

Captures d'écran :

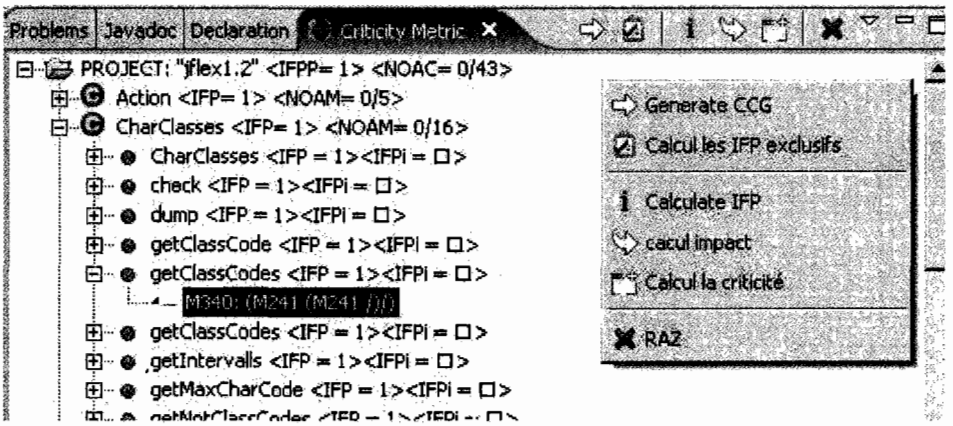


Figure 5 : GUI Calcul de CCG

Diagramme de séquence :

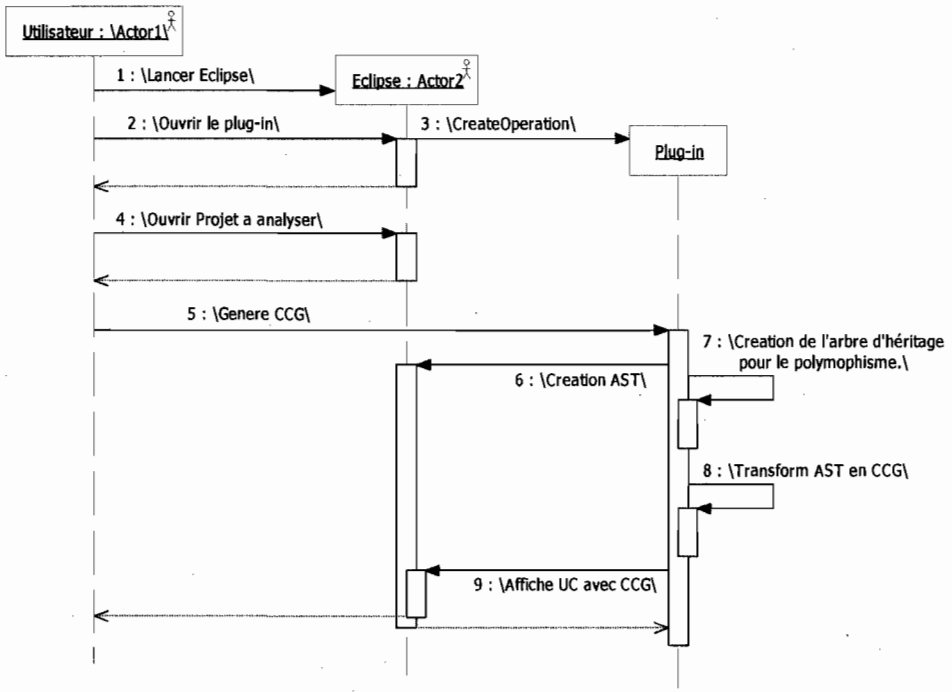


Figure 6: DS Calcul de CCG

Titre du cas d'utilisation :

Modifier le taux de couverture de test

Description sommaire :

L'utilisateur veut modifier le taux de couverture de test d'une ou de plusieurs méthodes ou classes (à des fins de maintenance). Par défaut, le système affecte une valeur de 1 à toutes les méthodes. La valeur de 1 signifie que la méthode est complètement testée.

Règles d'initiation :

- L'utilisateur doit avoir au préalable réalisé le cas d'utilisation 1
- L'utilisateur doit choisir en cliquant dessus, une méthode, ou une classe.

Description du processus :

1 L'utilisateur double-clique sur la méthode ou la classe choisie

2 L'utilisateur change le taux de couverture de test et applique soit à toutes les méthodes du logiciel, soit à toutes méthodes de la classe choisie, ou juste à la méthode choisie.

3 L'utilisateur applique son choix.

4 Le plug-in affecte et met à jour les nouveaux taux de couverture des éléments concernés.

Règles de terminaison :

- Le plug-in recalcule et met à jour les valeurs des IFPi et l'affichage.

Captures d'écran :

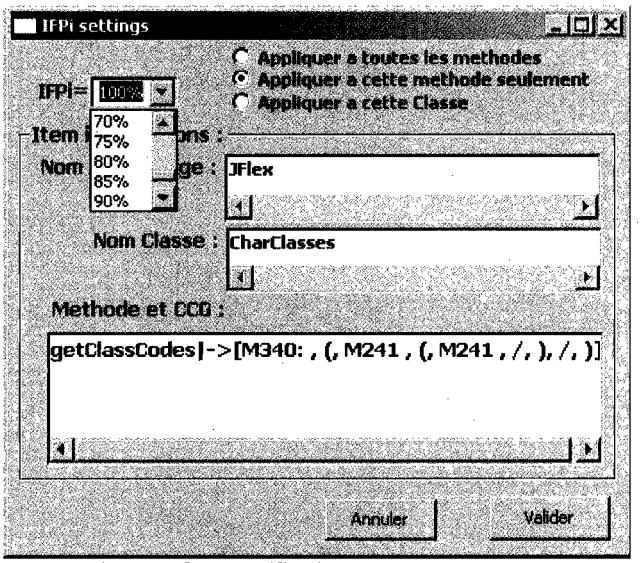


Figure 7: GUI Modification de couverture de test

Diagramme de séquence :

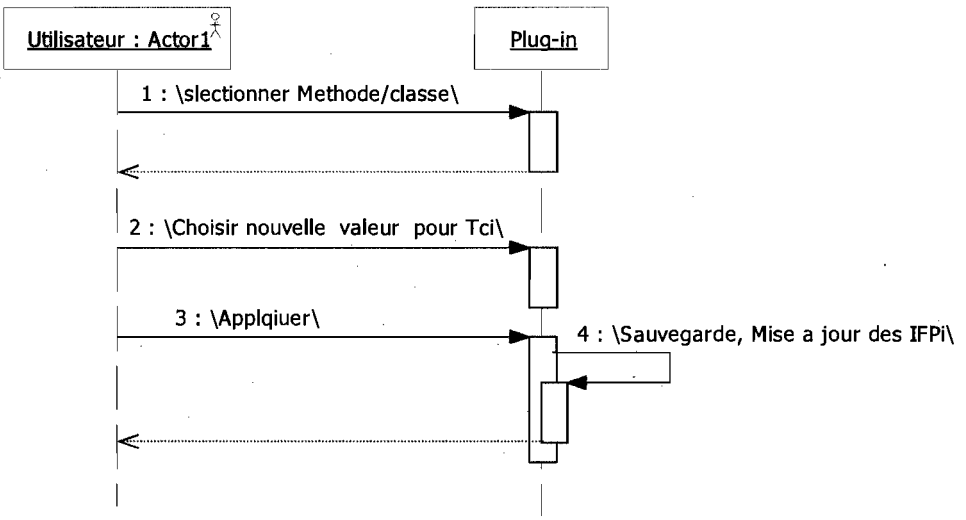


Figure 8: DS Modification de taux de couverture de test

Titre du cas d'utilisation :

Calcul des IFP (Qi) pour tous les composants

Description sommaire :

L'utilisateur veut calculer les IFP des méthodes, les Qi des classes et des paquetages après avoir modifié le taux de couverture d'une ou de plusieurs classes (cela peut-être a des fins d'analyse d'impact par exemple).

Règles d'initiation :

- L'utilisateur doit avoir au préalable réalisé le cas d'utilisation 2

Description du processus :

1 L'utilisateur clique sur « Calculer IFP »
2 Le plug-in convertit les CCG existants en système d'équations.
3 Le plug-in journalise le système d'équations sous forme de fonction Scilab dans un fichier.
4 Le plug-in calcule le nombre d'itérations nécessaires pour une erreur <10 – 5 et crée le scripte pour approximer la solution.
5 Le plugin initialise Scilab et lui fait exécuter le script.
6 Scilab, grâce au scripte, résout le système.
7 Le plug-in récupère les résultats, et calcule les Qi des classes et des unités de compilation.
8 Le plug-in journalise les résultats sous forme de fichier Excel
9 Le plug-in met à jour les résultats des IFP dans l'affichage de l'arbre.
10 Un message de fin de traitement est affiché.

Règles de terminaison :

- Le plug-in génère 3 fichiers Excel : un contenant les méthodes et leur IFP, un autre, les classes et leur Qi, et un autre encore pour les unités de compilation et les valeurs de leurs Qi.
- L'affichage de l'arbre doit être mis à jour.

Captures d'écran :

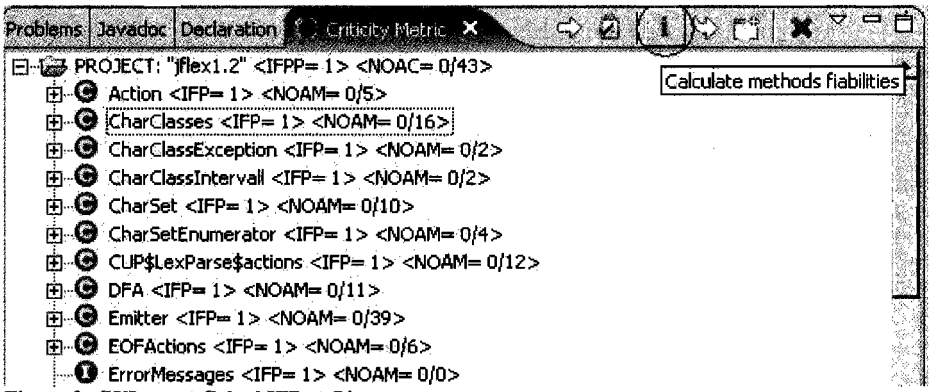


Figure 9: GUI avant Calcul IFP et Qi

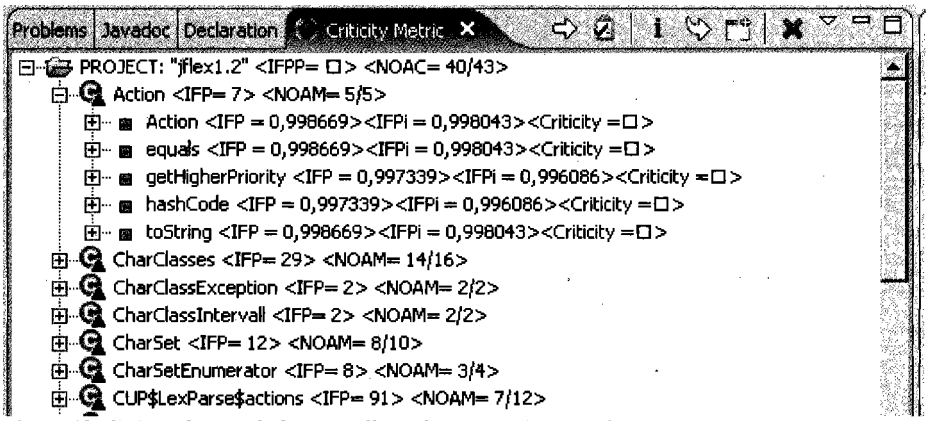


Figure 10: GUI après calcul, des nouvelles valeurs pour IFP et Qi

	A	B	C
1	**Package**	**Methodes**	**IFPs**
2	JFlex.CharClassException	CharClassException	0.998669276
3	JFlex.CharClassException	CharClassException	0.998669276
4	JFlex.StateSetEnumerator	StateSetEnumerator	0.998669276
5	JFlex.StateSetEnumerator	StateSetEnumerator	0.994547162
6	JFlex.StateSetEnumerator	reset	0.992015656
7	JFlex.StateSetEnumerator	advance	0.991009002
8	JFlex.StateSetEnumerator	hasMoreElements	0.997584344
9	JFlex.StateSetEnumerator	nextElement	0.992376505
10	java_cup.runtime.virtual_parse_stack	virtual_parse_stack	0.996812524
11	java_cup.runtime.virtual_parse_stack	get_from_real	0.997338552
12	java_cup.runtime.virtual_parse_stack	empty	0.998669276
13	java_cup.runtime.virtual_parse_stack	top	0.997338552

Figure 11: Fichier journalisation

Diagramme de séquence :

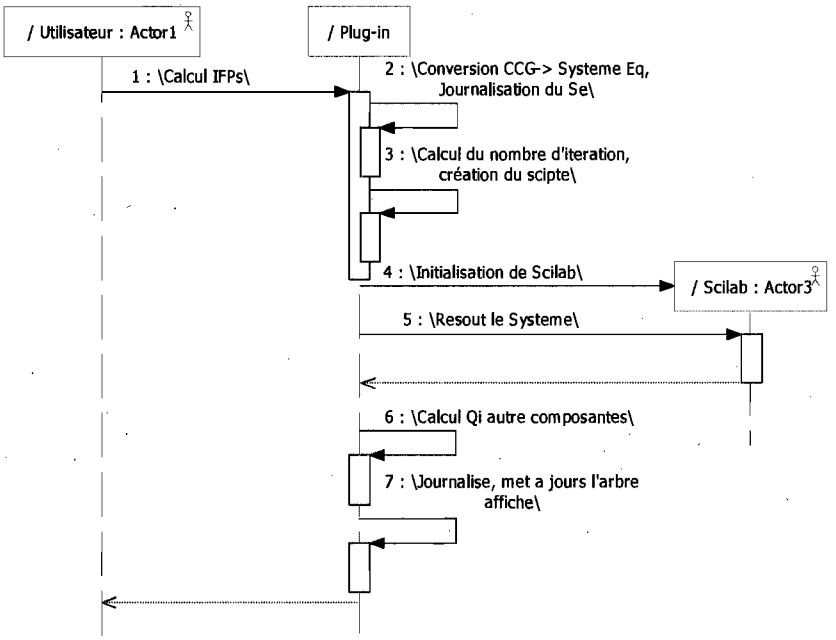


Figure 12: DS Calcul IFP

Titre du cas d'utilisation:

Calcul de la criticité des composants (Méthodes et classes).

Description sommaire:

L'utilisateur veut calculer la criticité d'un composant par rapport au système. Le calcul de criticité des composants peut nous renseigner sur l'importance de chaque composante (méthode ou classe) par rapport au système. Il s'agit plus d'importance relative dans le système qui permet, entre autres, de connaître les composants vis-à-vis desquels le système dans sa globalité est le plus sensible.

Règles d'initiation:

- L'utilisateur doit avoir au préalable réalisé le cas d'utilisation 1

Description du processus :

1 L'utilisateur clique sur « Calculer Criticité »

2 Le plug-in convertit le CCG existant en système d'équations.

3 Le plug-in baisse le taux de couverture d'une méthode i à 95 %, et laisse les autres à 1.

4 Le plug-in journalise le Système d'équations sous forme de fonction Scilab dans un fichier.

5 Le plug-in calcule le nombre d'itérations nécessaires pour une erreur <10 – 5 et crée le scripte pour approximer la solution.

6 Le plug-in initialise Scilab et lui fait exécuter le script.

7 Scilab, grâce au scripte, résout le système.

8 Le plug-in récupère les résultats, et calcule les Qi et les criticités $\frac{\partial Qi(systeme)}{\partial IFP_i}$ des méthodes

9 Le plug-in remet les valeurs de taux de couverture à 1

10 Le plug-in répète les étapes 3 -9 pour toutes les méthodes du logiciel à analyser.

11 Le plug-in journalise les résultats sous forme de fichier Excel

12 Le plug-in met à jour l'affichage de l'arbre.

13 Un message de fin de traitement est affiché.

Règles de terminaison :

- Le plug-in génère un fichier Excel contenant les méthodes et leur criticité par rapport au système.
- L'affichage de l'arbre doit être mis à jour.

Captures d'écran :

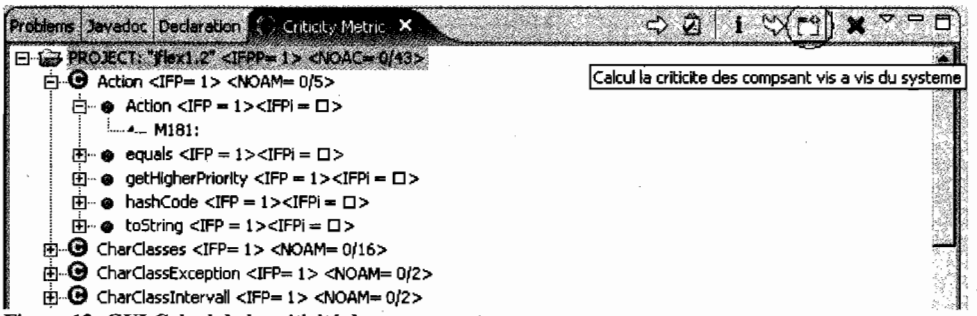


Figure 13: GUI Calcul de la criticité des composants

Diagramme de séquence :

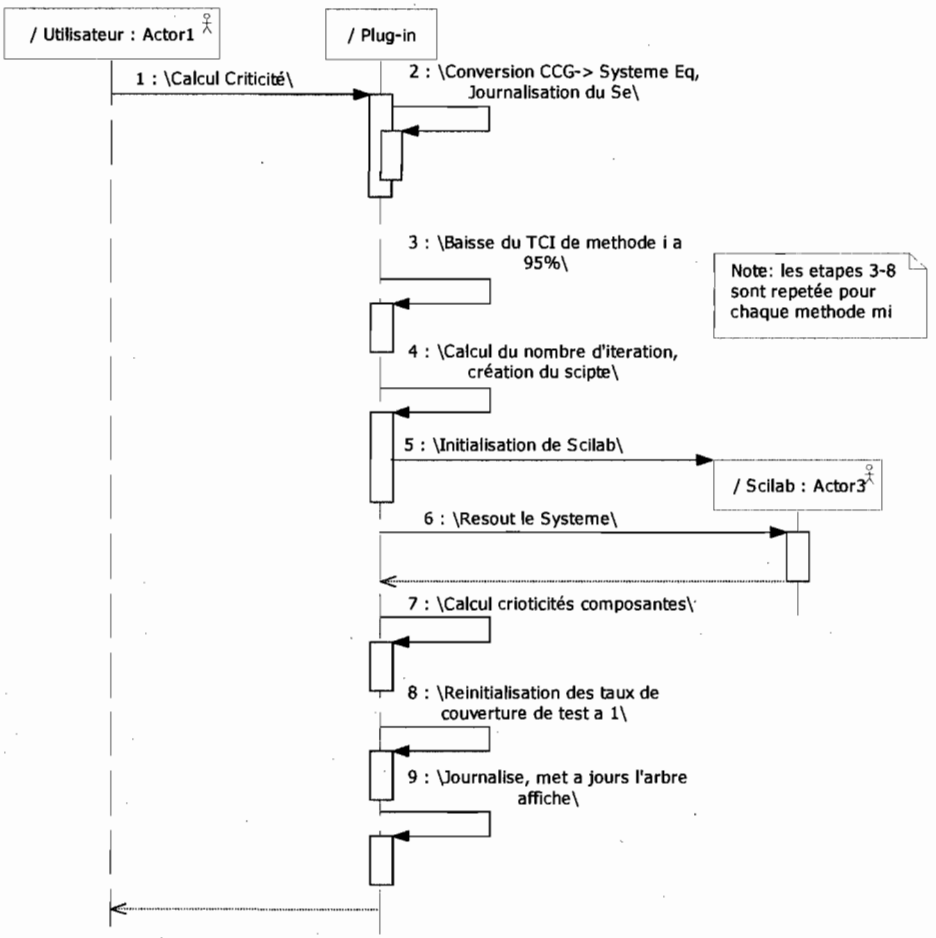


Figure 14:DS Calcul de la criticité des composants

Titre du cas d'utilisation:

Calcul de l'impact de changement d'une ou plusieurs méthodes sur le reste du système.

Description sommaire :

Il suffit de modifier juste la couverture de test des méthodes ayant subi les changements comme dans les cas d'utilisation : « Modifier le taux de couverture de test ». Ensuite, il faut recalculer les IFPs selon le cas d'utilisation : « Calcul des IFP et Qi des composants ». L'ensemble impact est constitué de l'ensemble des méthodes (respectivement classes) dont les IFP (respectivement Qi) ont changé.

4.6 Modèle du domaine

Ce modèle nous donne une vue des classes conceptuelles retenues lors de notre analyse de domaine.

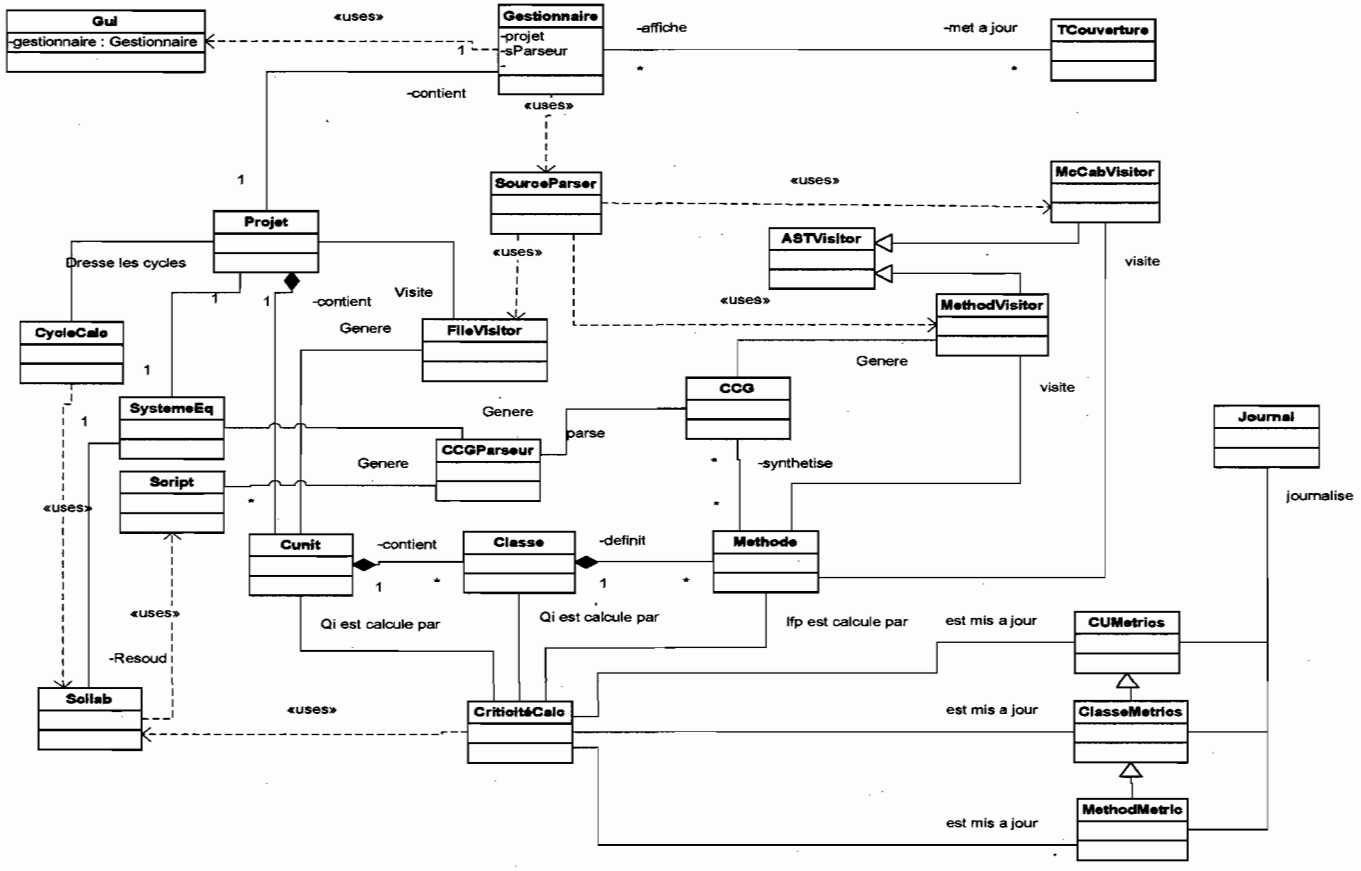


Figure 15 : Modèle du domaine

4.7 Spécifications supplémentaires

4.7.1 Performances

Concernant les performances de notre plug-in, ce dernier devra supporter une analyse de projet pouvant contenir 1500 classes, et cela, dans un délai raisonnable. Il devra aussi signaler et ignorer toute méthode non analysée, en justifiant les raisons. L'analyse concerne 1 projet à la fois. Les principales taches qu'effectuera le système sont reliées au calcul des Qi.

4.7.2 Base de données

Les métriques des classes sont stockées dans des fichiers EXCEL pour en faciliter le traitement. Il s'agit de 3 fichiers de journalisation nommés :

- ClassesIfp.xls

Chaque ligne de ce fichier contient l'information sur la valeur des indicateurs de qualité d'une classe repartie sur 5 colonnes. Le nom paquetage contenant la classe, le nom de la classe, la valeur moyenne des indicateurs des méthodes qu'elle contient, la valeur de Qi, ainsi que la moyenne des complexités cyclomatiques de ses méthodes.

- MethodesIfp.xls

Chaque ligne de ce fichier représente une méthode du système, pour bien l'identifier, les deux premières colonnes contiennent respectivement le nom du paquetage et le nom de la classe qui la contiennent, celle qui suit contient le nom de la méthode, et les deux autres colonnes donnent les valeurs des indicateurs et complexité cyclomatique de la méthode.

- CUnitIfp.xls

Ce fichier contient les unités de compilation, (fichier java) ainsi que la valeur de leur Qi calculées comme produit des Qi des classes qu'elle contient. Elle est identifiée par le paquetage qui la contient dans la première colonne.

- Cycles.txt

Ce fichier texte contient l'ensemble des cycles effectifs détectés par le module de calcul de cycles, si celui-ci n'est pas interrogé, ce fichier ne sera pas généré.

- Criticity.xls

Ce fichier contient la valeur de la criticité de chaque classe par rapport au système. Chaque classe est identifiée de manière unique par son paquetage.

Deux autres fichiers permettant de transférer le calcul à Scilab sont générés, il s'agit de :

- SystemEq.sci

C'est un fichier qui contient la fonction F (en langage Scilab) posée dans la partie [**3.1.3**].

- ScriptSolveur.sci

C'est un fichier script Scilab capable de résoudre le problème du point fixe de F par l'algorithme des approximations successives.

4.7.3 Exigences non fonctionnelles

La solution du problème du point fixe doit être exacte à 5 décimales prés afin d'avoir une meilleure capacité de discernement concernant les valeurs des indicateurs des méthodes, des classes et des unités de compilation.

CHAPITRE 5

EXPÉRIMENTATIONS

Dans ce chapitre, nous présentons les conditions générales dans lesquelles nous avons effectué nos expérimentations ainsi que les démarches que nous avons établies et réalisées pour valider les Qi en tant qu'indicateurs de certains attributs de qualité. Nous exposons, par la suite, d'autres protocoles expérimentaux qui serviront de base pour des travaux futurs. Les expérimentations, réalisées sur divers projets d'envergure, se regroupent en trois grandes familles visant à valider nos hypothèses de départ [section 1]. L'objectif consiste à ce niveau à :

- Démontrer avec des moyens statistiques l'existence de recoupements entre les métriques ainsi que l'unification qu'apportent les indicateurs de qualité (Qi) : Il s'agit du protocole d'expérimentation mis en place pour l'étude du recoupement entre métriques.

- Mettre en évidence la capacité de prédiction des changements des Qi (niveau classe) à l'aide d'une batterie d'expérimentations, et cela en comparaison avec d'autres métriques existantes : Ces protocoles se regroupent dans l'expérimentation sur la changeabilité.

- Monter à quel point les Qi constituent une mesure de l'effort à fournir pour effectuer le test d'une classe, en comparaison à d'autres métriques : Cela s'inscrit dans le cadre de l'expérimentation sur la testabilité.

- Exposer un modèle et un protocole d'analyse de l'impact de changements basé sur les indicateurs de qualité : c'est la famille d'expérimentations liées à l'analyse de l'impact du changement.

5.1 Conditions générales de l'expérimentation

Pour effectuer nos expérimentations, nous avons utilisé des outils statistiques ainsi que des environnements de développement intégrant des utilitaires de calcul des métriques logicielles. Pour nos expérimentations, nous avons eu à comparer les indicateurs de qualité à quelques métriques objets, définies dans le chapitre 1. Ces métriques sont calculées par l'outil Together 2006 de Borland. C'est un environnement implémentant les principales métriques connues et utilisées dans la technologie objet.

Les Qi sont calculés par notre plug-in, pour un taux de couverture de test de 75 % à toutes les méthodes des différents logiciels. Les traitements statistiques de données sont effectués grâce à l'outil XLSTAT 2007 [http://www.xlstat.com/fr/home/] développé par Addinsoft, qui dresse un rapport de l'analyse et suggère quelques conclusions. Nous effectuons ensuite une interprétation plus poussée des résultats.

5.2 Recoupement et unification des métriques

5.2.1 Problématique

La multiplicité des métriques peut cacher des redondances importantes quant aux attributs qu'elles sont censées capturer. L'objectif dans ce contexte, consiste à étudier dans quelles proportions ces dernières se chevauchent, et de trouver une manière de regrouper les métriques qui se recoupent en une seule. Le regroupement nous évitera le choix délicat de la métrique adéquate pour l'analyse d'un attribut de qualité donné.

5.2.2 Objectifs

Le protocole de cette première série d'expérimentations a pour but de déterminer en premier lieu les métriques capturées par les indicateurs de qualité. Pour cela, nous effectuons une analyse par composante principale (ACP) d'un ensemble de métriques incluant les Qi, et nous déterminons l'apport de chaque métrique dans la nouvelle base réduite. À l'aide de cet apport, nous pouvons calculer le taux de recoupement de chaque métrique sur l'autre. Pour les Qi en particulier, nous obtenons un pourcentage de capture des métriques qu'il englobe.

5.2.3 Démarche :

1- Choix de plusieurs (ou versions différentes) logiciels
2- Calcul des Qi des classes des logiciels
3- Calcul des métriques OO (choisies au préalable) de toutes les classes.
4- Analyse ACP sur l'ensemble des métriques, y compris les Qi.
5- Analyse des premières composantes qui couvrent plus de 95 % de l'information
6- Détermination des redondances.

5.2.4 Environnement et collecte des données

Nous avons choisi un grand nombre de logiciels open source, allant du logiciel d'analyse lexicographique (grande complexité), aux logiciels graphiques 3D (design sophistiqué) en passant par des logiciels Web, des logiciels orientés réseaux. Ce qui a pour effet de confronter notre analyse à une variété de cycle de développement, de programmeurs, et de domaines informatiques différents.

Sept (7) logiciels écrits en Java ont été choisis, en plusieurs versions pour certains. Il s'agit de : JFlex, JMol, FreeCs, GnuJSP, Oro, Lucene, Snark.

- JFlex : [http://jflex.de/]

C'est un analyseur lexical et générateur de code source Java. Il reçoit en entrée des spécifications à partir d'un fichier, dans un format spécial, et génère le code source Java correspondant. Il a été développé par Elliot BERK de l'université de Princeton. Son code source est ouvert au public. Nous avons choisi les 13 premières versions de JFlex.

- JMol : [http://jmol.sourceforge.net/]

JMol est un logiciel gratuit de visualisation de molécules, open source pour les étudiants, les éducateurs et les chercheurs en chimie et en biochimie. Il est multiplateforme, fonctionnant sous Windows, Mac OS X et les systèmes Linux/Unix. JMol a commencé en tant que projet OpenScience (http:// www.openscience.org) qui est dédié à l'écriture et la livraison de logiciels scientifiques gratuits et open source. Pour nos analyses, nous avons choisi 9 versions de ce dernier.

- GnuJSP v1.0.1: [http://www.klomp.org/gnujsp/]

C'est une implémentation open source gratuite du serveur JSP de Sun. Elle a été développée initialement par Vincent Partington, puis reprise par une communauté de développeurs à travers internet.

- Oro v 2.0.8, Lucene v 2.2.0: [http://www.apache.org]

Oro est une librairie de traitement de texte écrite en Java, tandis que Lucene est une libraire complète qui permet des recherches textuelles. Ces deux libraires sont utilisées pour toute application nécessitant des outils de traitement de texte. Leur développement est réalisé grâce à une importante communauté de développeurs et est supporté par « Apache Software Foundation ».

- Snark v 0.5 : [http://www.klomp.org/snark/]

Il s'agit d'un client pour le téléchargement et le partage de fichiers pour le protocole de Bittorent. Il est principalement utilisé pour explorer ce protocole et

son expérimentation avec le compilateur de Java de GNU (gcj). Il peut aussi être utilisé comme un client BitTorrent simple.

- FreeCs v1.2.2 : [http://freecs.sourceforge.net/]

FreeCs est un serveur de discussion en ligne (chat server) écrit en Java. Il supporte une interface entièrement personnalisable aussi bien du côté client que du côté serveur.

Les outils utilisés pour les calculs des métriques sont décrits dans les conditions générales d'expérimentation. Le choix des métriques s'est fait parmi celles citées dans le chapitre 2, car elles ont été validées dans d'autres travaux antérieurs en tant métriques associées à certains attributs que nos indicateurs de qualité sont supposés capturer. Nous avons ajouté deux métriques supplémentaires il s'agit de :

- Cohésion étroite entre classes (TCC) [Bieman95] qui se défini comme le nombre relatif de méthodes directement connectées. Deux méthodes sont directement connectées si elles accèdent à une variable d'instance commune de la classe. Dans une classe *c*, soit *n* le nombre de méthode : alors *NP(c)* le nombre de paires de méthodes de la classe est donné par :

$$NP(c) = \frac{n(n-1)}{2}$$

Soit *ND(c)* le nombre de paires de méthodes de la classe qui accèdent directement à une même variable membre de la classe. La cohésion étroite *TCC* est alors donnée par la formule suivante :

$$TCC(c) = \frac{NP(c)}{ND(c)} = \frac{n(n-1)}{2 * ND(c)}$$

Le TCC est une mesure de cohésion normalisée. Une grande cohésion dans une classe est souhaitée, car elle reflète une bonne modélisation et rejoint la définition du concept de classe comme étant une unité structurelle dans laquelle les méthodes travaillent ensembles avec les variables membres communes.

- **Le Couplage par invocation de méthode (MIC)**[*] qui est le nombre (relatif) des autres classes auxquelles une certaine classe envoie des messages.

$$MIC_{norm} = \frac{n_{MIC}}{(N-1)}$$

où N correspond au nombre total des classes définies dans le système et n_{mic} au nombre de classes auxquelles des messages sont envoyés. Le MIC a un impact sur certains attributs externes de la qualité tels que :

- Facilité de maintenance : La maintenance d'une classe fortement couplée (valeur MIC élevée) est plus difficile à assurer en raison de sa dépendance vis-à-vis des classes auxquelles elle est couplée.

- Compréhensibilité : La compréhension d'une classe fortement couplée est plus difficile, car elle implique la compréhension partielle (et parfois totale) des classes auxquelles elle est couplée par invocation de méthode.

- Propagation des erreurs et testabilité. Le nombre d'erreurs se produisant dans une classe est directement proportionnel au nombre de couples avec d'autres classes. Par conséquent, un niveau de couplage élevé a un impact négatif sur la testabilité. La définition proposée pour MIC est normalisée. Les avantages sont certains, mais de plusieurs points de vue (notamment celui de la facilité de maintenance), il est plus important d'utiliser des valeurs absolues, c'est-à-dire le nombre de classes auxquelles la classe est couplée. Pour certains points de vue, il peut

[*] Implémenté dans Borland Together 2006

s'avérer important de compter uniquement les couplages du système avec les classes définies par l'utilisateur, c.-à-d. d'exclure les classes de bibliothèque.

5.2.5 Statistiques descriptives générales des systèmes analysés

Le tableau 2 donne quelques statistiques générales de l'ensemble de systèmes que nous avons analysés. Les 29 systèmes analysés ont un total de 400 000 lignes de code environ et englobent près de 4000 classes et 21 600 méthodes. Le système d'héritage est aussi très présent dans notre échantillon, en effet 8.7 % de classes (soit environ 340) héritent d'une classe mère autre que la classe Object. La présence du système d'héritage permettra de vérifier que nos indicateurs de qualité capturent le polymorphisme (caractéristique importante des systèmes orientés objet). Le nombre de variables publiques/privées reflète le degré d'encapsulation des données. Plus de variables privées rendent le flux de contrôle beaucoup plus important que le flux de donnée, et donc nos indicateurs de qualité deviennent plus précis avec l'encapsulation.

Le tableau 3 donne une description statistique (Minimum, maximum, Moyenne, écart-type) des métriques utilisées pour les différents logiciels; rappelons que pour JMOL et JFLEX, il s'agit de la concaténation de 9 respectivement 14 versions des logiciels.

Notons aussi que le nombre d'observations correspond au nombre d'unités de compilation (de fichier.java) et non le nombre de classes.

Logiciel	LOC	Classes	Méth	% VPbl	%VPrv	%Cl-Fille
FreeCS	15141	128	848	49,69%	50,31%	37,50%
Oro2.0.8	6642	86	353	11,36%	86,36%	11,63%
Gnujsp-1.0.1	5136	81	463	0,00%	97,48%	18,52%
Lucene2.2.0	22940	322	1859	6,24%	79,43%	7,45%
Snark-05	4793	33	243	0,95%	99,05%	6,06%
Jflex1.2	7240	43	319	15,32%	72,97%	9,30%
Jflex1.2.1	7277	43	321	15,18%	73,21%	9,30%
Jflex1.2.2	7467	44	325	15,04%	73,45%	9,09%
Jflex1.3	8298	45	339	15,83%	73,33%	11,11%
Jflex1.3.1	8447	45	347	17,21%	72,13%	11,11%
Jflex1.3.2	8519	45	347	17,21%	72,13%	11,11%
Jflex1.3.3	9184	49	379	16,91%	73,53%	10,20%
Jflex1.3.4	9232	49	380	16,91%	73,53%	10,20%
Jflex1.3.5	9272	49	380	16,91%	73,53%	10,20%
Jflex1.4pre1	9545	48	385	16,91%	73,53%	10,42%
Jflex1.4pre3	9702	48	390	16,91%	73,53%	10,42%
Jflex1.4pre4	9887	60	477	39,15%	54,04%	3,33%
Jflex1.4pre5	9887	60	477	39,15%	54,04%	3,33%
Jmol1	18357	237	1073	13,97%	79,47%	5,91%
Jmol1.1	20228	256	1153	19,67%	74,24%	5,47%
Jmol1.2	22548	259	1164	19,60%	74,34%	5,41%
Jmol2	18166	206	950	9,63%	85,19%	7,28%
Jmol3	18666	214	996	9,07%	83,57%	7,48%
Jmol4	19324	225	956	9,10%	82,56%	8,44%
Jmol5	21908	249	1079	8,40%	83,90%	8,03%
Jmol6	28252	293	1656	24,01%	70,16%	7,51%
Jmol7	29650	313	1718	23,67%	69,99%	7,67%
Jmol8	30288	315	1732	23,21%	70,54%	7,94%
Total	406015	3905	21587			

Tableau 2:Statistiques descriptives des projets analysés

Projets	Métriques	No	Min	Max	Moyenne	E-type
	Qi	663	0.172	1	0.943	0.123
	CBO	663	0.000	29	5.112	6.803
	DOIH	663	0.000	5	1.246	0.929
	LCOM1	663	0.000	674	37.535	119.474
JFLEX	LOC	663	2.000	2539	166.192	360.892
	MIC	663	0.000	26	3.919	5.936
	MPC	663	0.000	223	25.231	39.821
	NOO	663	0.000	48	7.821	9.741
	RFC	663	0.000	435	45.744	79.557
	TCC	663	0.000	100	12.671	19.142
	WMPC1	663	0.000	234.000	24.754	39.884
	Qi	1433	0.013	1	0.807	0.228
	CBO	1433	0.000	104	11.104	13.835
	DOIH	1433	0.000	7	2.264	1.867
	LCOM1	1433	0.000	32168	121.218	1400.624
JMOL	LOC	1433	3.000	2275	151.167	252.047
	MIC	1433	0.000	20	1.767	2.378
	MPC	1433	0.000	426	40.231	62.825
	NOO	1433	0.000	283	8.500	16.443
	RFC	1433	1.000	627	130.263	180.098
	TCC	1433	0.000	100	11.514	18.811
	WMPC1	1433	1.000	317	21.425	29.238
	Qi	121	0.038	1	0.723	0.216
	CBO	121	0.000	61	8.942	8.454
	DOIH	121	0.000	4	1.471	0.720
	LCOM1	121	0.000	3514	62.521	343.106
FREECS	LOC	121	3.000	872	125.132	163.997
	MIC	121	0.000	43	3.529	4.585
	NOO	121	0.000	91	7.008	11.220
	MPC	121	0.000	235	33.851	44.097
	RFC	121	0.000	177	48.488	34.355
	TCC	121	0.000	100	5.314	14.018
	WMPC1	121	0.000	202	26.281	36.091
	Qi	77	0.006	1	0.819	0.268
	CBO	77	0.000	37	3.688	6.582
	DOIH	77	0.000	4	1.519	0.940
	LCOM1	77	0.000	420	24.883	69.203
GNUJSP	LOC	77	5.000	936	66.701	144.418

	MIC	77	0.000	11	0.922	2.138
	NOO	77	0.000	38	6.013	7.993
	MPC	77	0.000	289	17.610	43.531
	RFC	77	0.000	153	32.026	29.427
	TCC	77	0.000	100	45.506	47.260
	WMPC1	77	0.000	138.000	13.299	23.773
LUCENE	Qi	234	0.173	1	0.939	0.110
	CBO	234	0.000	45	5.897	6.801
	DOIH	234	0.000	5	1.701	1.004
	LCOM1	234	0.000	2445	42.248	224.377
	LOC	234	3.000	1223	95.872	153.182
	MIC	234	0.000	10	0.906	1.423
	NOO	234	0.000	69	7.761	8.954
	MPC	234	0.000	253	22.043	32.137
	RFC	234	0.000	180	34.547	27.274
	TCC	234	0.000	100	10.201	17.287
	WMPC1	234	0.000	282	20.782	34.098
ORO	Qi	81	0.098	1	0.904	0.173
	CBO	81	0.000	25	4.753	5.081
	DOIH	81	0.000	5	1.407	1.010
	LCOM1	81	0.000	186	11.654	33.322
	LOC	81	3.000	1253	81.235	195.181
	MIC	81	0.000	15	1.988	2.939
	NOO	81	0.000	26	4.346	6.075
	MPC	81	0.000	146	13.840	23.387
	NOOM	81	0.000	11	1.519	2.550
	RFC	81	1.000	358	28.852	40.131
	TCC	81	0.000	100	8.358	18.949
	WMPC1	81	0.000	266	15.383	39.252
SNARK	Qi	33	0.060	1	0.540	0.299
	CBO	33	0.000	31	8.091	6.597
	DOIH	33	0.000	4	1.091	0.723
	LCOM1	33	0.000	76	11.636	21.697
	LOC	33	4.000	481	145.242	126.838
	MIC	33	0.000	22	8.121	7.474
	NOO	33	0.000	24	7.364	6.599
	MPC	33	0.000	99	29.879	29.707
	RFC	33	1.000	88	37.424	23.914
	TCC	33	0.000	100	14.152	25.472
	WMPC1	33	1.000	87	22.818	22.442

Tableau 3:Statistiques descriptives des métriques des systèmes

5.2.6 L'approche statistique : Analyse par Composantes Principales

L'Analyse par Composantes Principales (ACP) est l'une des méthodes d'analyse de données multi variées les plus utilisées. Dès lors que l'on dispose d'un tableau de données quantitatives (continues ou discrètes) dans lequel n observations (n classes) sont décrites par p variables (nos métriques...), si p est assez élevé, il est impossible d'appréhender la structure des données et la proximité entre les observations en se contentant d'analyser des statistiques descriptives uni variées ou même une matrice de corrélation. Il existe plusieurs applications pour l'ACP, parmi lesquelles :

- l'étude et la visualisation des corrélations entre les variables, afin de limiter éventuellement le nombre de variables à mesurer par la suite (notre objectif) ;

- l'obtention de facteurs non corrélés qui sont des combinaisons linéaires des variables de départ, afin d'utiliser ces facteurs dans des méthodes de modélisation telles que la régression linéaire, la régression logistique ou l'analyse discriminante;

- la visualisation des observations dans un espace à deux ou trois dimensions, afin d'identifier des groupes homogènes d'observations, ou au contraire des observations atypiques.

L'ACP repose sur un principe de projection. Elle projette les observations depuis l'espace à p dimensions des p variables (nos p métriques) vers un espace à k dimensions ($k<p$) tel qu'un maximum d'informations soit conservé (l'information est ici mesurée au travers de la variance totale du nuage de points) sur les premières dimensions. Si l'information associée aux 2 ou 3 premiers axes représente un pourcentage suffisant de la variabilité totale du nuage de points, on pourra représenter les observations sur un graphique à 2 ou

3 dimensions, facilitant ainsi grandement l'interprétation. Ceci est un élément important dans la métrologie logicielle car il permet de déterminer les éléments de base fournissant l'information la plus importante facilitant la gestion et l'interprétation des résultats fournis par les métriques.

L'ACP utilise une matrice indiquant le degré de similarité entre les variables pour calculer des matrices permettant la projection des variables dans le nouvel espace. Il est commun d'utiliser comme indice de similarité le coefficient de corrélation de Pearson, ou la covariance. La corrélation de Pearson et la covariance présentent l'avantage de donner des matrices semi-définies positives dont les propriétés sont utilisées en ACP. Néanmoins, on peut envisager d'utiliser d'autres indices. Nous utiliserons la corrélation de Pearson.

Classiquement, on utilise un coefficient de corrélation et non la covariance, car l'utilisation du coefficient de corrélation permet de supprimer les effets d'échelle: ainsi, une variable variant entre 0 et 1 ne pèse pas plus dans la projection qu'une variable variant entre 0 et 1000. Toutefois, dans certains domaines, lorsque les variables sont supposées être sur des échelles identiques, ou lorsque l'on veut que la variance des variables influe sur la construction des facteurs, on utilise la covariance.

Après l'analyse par composante principale, notre approche consistera à calculer l'apport global de chaque métrique sur les 5 ou 6 premières composantes qui représentent 95% de l'information. Nous calculerons le pourcentage de capture de chaque métrique par les Qi.

En considérant les moyennes des indicateurs tels que la taille (LOC), la cohésion (LCOM), le couplage (CBO) et l'héritage (DOIH), on fait les constats suivants :

- Les plus fortes valeurs du couplage (au sens de CBO) sont pour les logiciels JMOL, FREECS, et SNARK avec 11.104, 8.454, 8.09, les logiciels restants présentent des valeurs moyennes plus petites que 6.0. Le couplage est par ailleurs faible pour GNUJSP et ORO avec 3.69 et 4.75.

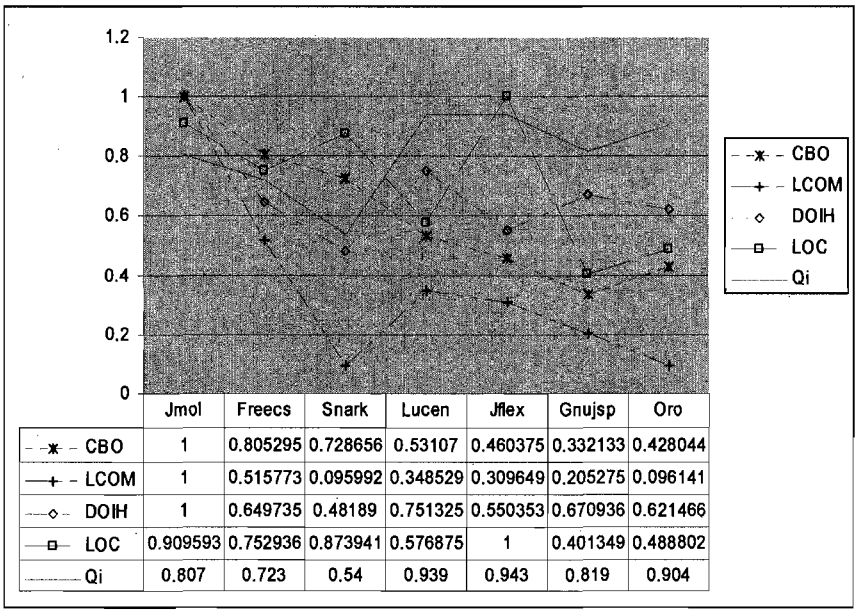
- Les fortes cohésions au sens de LCOM (LCOM faible) sont observées pour SNARK et ORO ; 11.64 et 11.65 respectivement. Notons aussi que les plus faibles cohésions sont relevées au niveau de JMOL 121.22 et FREECS avec 62.52.

- La taille moyenne des unités de compilation varie entre 81.23 pour ORO et 166.19 lignes de code par fichier (.java) pour JFLEX.

- L'utilisation de l'héritage est beaucoup plus prononcée pour JMOL avec une moyenne de 2.264; néanmoins, il reste sensiblement égal pour les autres logiciels où les la plus faible valeur est observée pour SNARK 1.09.

- Pour les indicateurs de qualité, les valeurs moyennes sont comprises entre 0.54 pour SNARK et 0.94 pour JFLEX.

La figure 16 résume ces commentaires. Elle montre les tendances des métriques de base de Chidamber & Kemerer [Chidamber 94] en plus de celles des indicateurs de qualité.



	Jmol	Freecs	Snark	Lucen	Jflex	Gnujsp	Oro
CBO	1	0.805295	0.728656	0.53107	0.460375	0.332133	0.428044
LCOM	1	0.515773	0.095992	0.348529	0.309649	0.205275	0.096141
DOIH	1	0.649735	0.48189	0.751325	0.550353	0.670936	0.621466
LOC	0.909593	0.752936	0.873941	0.576875	1	0.401349	0.488802
Qi	0.807	0.723	0.54	0.939	0.943	0.819	0.904

Figure 16: Description moyenne normalisée des tendances des principales métriques des 7 systèmes

5.2.7 Résultats et interprétations

5.2.7.1 Matrice de corrélations de Pearson

Les valeurs en **gras** dans les tableaux suivants sont significativement différentes de 0 avec un degré de signification alpha = 0.05 pour toutes les matrices de corrélation. Notons aussi que les signes négatifs des corrélations ont tout leur sens dans ces matrices. Les Qi sont des indicateurs de qualité, et nous nous attendons à ce qu'une grande valeur de Qi (proche de 1) soit signe de bonne conception pour un logiciel. Les signes de corrélations montrent tous que les Qi baissent avec :

- l'augmentation du couplage (sens CBO, MIC),
- la baisse de la cohésion (augmentation de LCOM1, diminution de TCC),
- l'augmentation de la profondeur de l'héritage (quant celui-ci est significatif, JMOL a le plus fort taux d'héritage de tous les projets analysés ; 2.264).

- l'augmentation de la complexité cyclomatique (WMPC1, RFC).
- L'augmentation de la taille (augmentation de LOC, NOO, RFC).

JFLEX : Le tableau 4 montre des corrélations significatives entre Qi et toutes les autres métriques, à l'exception de DOIH. La faible sensibilité de Qi par rapport au DOIH peut s'expliquer par la faible utilisation de l'héritage dans JFLEX (le deuxième plus faible après SNARK). Ce qui confirme qu'une utilisation modérée de l'héritage ne nuit pas à la qualité (pas de dégradation de Qi). Qi et LOC sont fortement corrélés. Le signe négatif de cette dernière montre que l'augmentation de lignes de code dégrade la valeur de Qi conformément aux hypothèses que nous avons énoncées précédemment.

Remarquons, par ailleurs, la forte et significative corrélation entre Qi et LCOM, une corrélation observée pour tous les systèmes. Dans notre modèle, les Qi ne prennent pas en compte les flux de donnée. De ce fait, il n'existe aucun lien direct entre la cohésion au sens de LCOM et les Qi. Pour expliquer cette corrélation, on suppose qu'il existe un lien entre LCOM et une des autres métriques qui est corrélées avec Qi. On pense notamment aux métriques de couplage (CBO, MPC, MIC).

Métriques	Qi	CBO	DOIH	LCOM1	LOC	MIC	MPC	NOO	RFC	TCC	WMPC1
Qi	**1**	**-0.545**	0.052	**-0.540**	**-0.922**	**-0.554**	**-0.557**	**-0.638**	**-0.159**	**0.147**	**-0.579**
CBO	**-0.545**	**1**	**0.236**	**0.387**	**0.609**	**0.867**	**0.812**	**0.660**	**0.568**	**-0.097**	**0.690**
DOIH	0.052	**0.236**	**1**	-0.043	-0.033	0.001	0.027	-0.048	**0.767**	-0.035	**-0.086**
LCOM1	**-0.540**	**0.387**	-0.043	**1**	**0.609**	**0.363**	**0.659**	**0.804**	**0.160**	**-0.170**	**0.711**
LOC	**-0.922**	**0.609**	-0.033	**0.609**	**1**	**0.570**	**0.640**	**0.696**	**0.206**	**-0.088**	**0.692**
MIC	**-0.554**	**0.867**	0.001	**0.363**	**0.570**	**1**	**0.784**	**0.621**	**0.269**	**-0.146**	**0.705**
MPC	**-0.557**	**0.812**	0.027	**0.659**	**0.640**	**0.784**	**1**	**0.883**	**0.372**	**-0.162**	**0.921**
NOO	**-0.638**	**0.660**	-0.048	**0.804**	**0.696**	**0.621**	**0.883**	**1**	**0.284**	**-0.175**	**0.924**
RFC	**-0.159**	**0.568**	**0.767**	**0.160**	**0.206**	**0.269**	**0.372**	**0.284**	**1**	-0.048	**0.219**
TCC	**0.147**	**-0.097**	-0.035	**-0.170**	**-0.088**	**-0.146**	**-0.162**	**-0.175**	-0.048	**1**	**-0.160**
WMPC1	**-0.579**	**0.690**	**-0.086**	**0.711**	**0.692**	**0.705**	**0.921**	**0.924**	**0.219**	**-0.160**	**1**

Tableau 4:Matrice de corrélation JFLEX

JMol [tableau5] est une application graphique où la conception est fortement basée sur un système d'héritage. Nos statistiques descriptives indiquent que c'est le système qui contient les structures d'héritage les plus profondes (2.264 en moyenne) parmi nos 7 projets. L'effet de cette structure sur la baisse Qi se fait remarquer par l'apparition d'une corrélation significative au sens alpha de -0.34. Le signe négatif indique une dégradation des Qi avec l'augmentation de DOIH.

Métriques	Qi	CBO	DOIH	LCOM1	LOC	MIC	MPC	NOO	RFC	TCC	WMPC1
Qi	1	**-0.632**	**-0.336**	**-0.226**	**-0.716**	**-0.493**	**-0.772**	**-0.561**	**-0.422**	**0.169**	**-0.693**
CBO	**-0.632**	1	**0.549**	**0.145**	**0.540**	**0.711**	**0.803**	**0.312**	**0.629**	**-0.096**	**0.388**
DOIH	**-0.336**	**0.549**	1	-0.028	**0.216**	**0.140**	**0.391**	**0.072**	**0.921**	**-0.061**	**0.066**
LCOM1	**-0.226**	**0.145**	-0.028	1	**0.226**	**0.227**	**0.295**	**0.826**	**0.104**	-0.047	**0.539**
LOC	**-0.716**	**0.540**	**0.216**	**0.226**	1	**0.398**	**0.881**	**0.464**	**0.332**	**-0.107**	**0.648**
MIC	**-0.493**	**0.711**	**0.140**	**0.227**	**0.398**	1	**0.584**	**0.322**	**0.230**	**-0.075**	**0.458**
MPC	**-0.772**	**0.803**	**0.391**	**0.295**	**0.881**	**0.584**	1	**0.533**	**0.519**	**-0.110**	**0.657**
NOO	**-0.561**	**0.312**	**0.072**	**0.826**	**0.464**	**0.322**	**0.533**	1	**0.225**	**-0.132**	**0.839**
RFC	**-0.422**	**0.629**	**0.921**	**0.104**	**0.332**	**0.230**	**0.519**	**0.225**	1	**-0.054**	**0.198**
TCC	**0.169**	**-0.096**	**-0.061**	-0.047	**-0.107**	**-0.075**	**-0.110**	**-0.132**	**-0.054**	1	**-0.144**
WMPC1	**-0.693**	**0.388**	**0.066**	**0.539**	**0.648**	**0.458**	**0.657**	**0.839**	**0.198**	**-0.144**	1

Tableau 5:Matrice de corrélation JMol

Freecs [tableau 6]: La corrélation non significative entre Qi et DOIH confirme que la faible utilisation de l'héritage (0.65) n'a pas d'incidence sur l'indicateur de qualité.

Métriques	Qi	CBO	DOIH	LCOM1	LOC	MIC	NOO	MPC	RFC	TCC	WMPC1
Qi	1	**-0.538**	-0.169	**-0.395**	**-0.769**	**-0.398**	**-0.570**	**-0.667**	**-0.690**	**0.179**	**-0.749**
CBO	**-0.538**	1	-0.019	**0.270**	**0.632**	**0.852**	**0.510**	**0.774**	**0.789**	0.016	**0.599**
DOIH	-0.169	-0.019	1	-0.099	-0.114	0.048	**-0.199**	-0.072	**0.191**	-0.142	-0.133
LCOM1	**-0.395**	**0.270**	-0.099	1	**0.500**	**0.179**	**0.870**	**0.469**	**0.489**	-0.042	**0.605**
LOC	**-0.769**	**0.632**	-0.114	**0.500**	1	**0.386**	**0.756**	**0.902**	**0.793**	-0.013	**0.946**
MIC	**-0.398**	**0.852**	0.048	**0.179**	**0.386**	1	**0.341**	**0.507**	**0.542**	-0.071	**0.353**
NOO	**-0.570**	**0.510**	**-0.199**	**0.870**	**0.756**	**0.341**	1	**0.749**	**0.719**	-0.023	**0.814**
MPC	**-0.667**	**0.774**	-0.072	**0.469**	**0.902**	**0.507**	**0.749**	1	**0.912**	0.045	**0.882**
RFC	**-0.690**	**0.789**	**0.191**	**0.489**	**0.793**	**0.542**	**0.719**	**0.912**	1	0.027	**0.783**
TCC	**0.179**	0.016	-0.142	-0.042	-0.013	-0.071	-0.023	0.045	0.027	1	-0.022
WMPC1	**-0.749**	**0.599**	-0.133	**0.605**	**0.946**	**0.353**	**0.814**	**0.882**	**0.783**	-0.022	1

Tableau 6:Matrice des corrélations de FREECS

GNUJSP [tableau 7] présente des corrélations significatives entre les Qi et les autres métriques sauf pour DOIH, encore une fois la structure d'héritage dans GNUSJP semble raisonnable à la lumière des statistiques descriptives dressées plus haut. Les Systèmes suivants présentent à peu près les mêmes résultats que GNUJSP. Cependant, on remarque pour ces systèmes que la corrélation entre Qi et TCC n'est pas significative, ce qui est difficile à interpréter, mais nous démontre que TCC et LCOM ne capturent pas la même dimension de la cohésion dans certains cas. Cette problématique a également été soulevée dans certains travaux de recherche sur la cohésion des systèmes OO.

Métriques	Qi	CBO	DOIH	LCOM1	LOC	MIC	NOO	MPC	RFC	TCC	WMPC1
Qi	**1**	**-0.796**	-0.121	**-0.767**	**-0.840**	**-0.801**	**-0.851**	**-0.844**	**-0.836**	**0.497**	**-0.915**
CBO	**-0.796**	**1**	0.216	**0.468**	**0.858**	**0.892**	**0.595**	**0.885**	**0.820**	**-0.496**	**0.749**
DOIH	-0.121	0.216	**1**	0.185	0.163	0.132	0.099	0.179	**0.484**	**-0.512**	0.099
LCOM1	**-0.767**	**0.468**	0.185	**1**	**0.588**	**0.540**	**0.731**	**0.599**	**0.652**	**-0.335**	**0.676**
LOC	**-0.840**	**0.858**	0.163	**0.588**	**1**	**0.867**	**0.621**	**0.984**	**0.822**	**-0.381**	**0.887**
MIC	**-0.801**	**0.892**	0.132	**0.540**	**0.867**	**1**	**0.507**	**0.892**	**0.723**	**-0.394**	**0.761**
NOO	**-0.851**	**0.595**	0.099	**0.731**	**0.621**	**0.507**	**1**	**0.624**	**0.773**	**-0.444**	**0.797**
MPC	**-0.844**	**0.885**	0.179	**0.599**	**0.984**	**0.892**	**0.624**	**1**	**0.841**	**-0.364**	**0.857**
RFC	**-0.836**	**0.820**	**0.484**	**0.652**	**0.822**	**0.723**	**0.773**	**0.841**	**1**	**-0.583**	**0.817**
TCC	**0.497**	**-0.496**	**-0.512**	**-0.335**	**-0.381**	**-0.394**	**-0.444**	**-0.364**	**-0.583**	**1**	**-0.419**
WMPC1	**-0.915**	**0.749**	0.099	**0.676**	**0.887**	**0.761**	**0.797**	**0.857**	**0.817**	**-0.419**	**1**

Tableau 7:Matrice des corrélations de GNUJSP

Nous donnons, dans ce qui suit, les tableaux des valeurs obtenues pour les autres systèmes analysés. Les conclusions sont sensiblement les mêmes que celles déjà discutées dans ce qui a précédé.

LUCENE :

Métriques	Qi	CBO	DOIH	LCOM1	LOC	MIC	NOO	MPC	RFC	TCC	WMPC1
Qi	**1**	**-0.604**	0.051	**-0.563**	**-0.920**	**-0.587**	**-0.737**	**-0.716**	**-0.654**	0.090	**-0.926**
CBO	**-0.604**	**1**	-0.005	**0.572**	**0.595**	**0.868**	**0.720**	**0.857**	**0.829**	-0.068	**0.553**
DOIH	0.051	-0.005	**1**	-0.084	-0.073	0.061	**-0.170**	-0.002	**0.196**	-0.010	-0.106
LCOM1	**-0.563**	**0.572**	-0.084	**1**	**0.589**	**0.555**	**0.796**	**0.695**	**0.587**	-0.089	**0.589**
LOC	**-0.920**	**0.595**	-0.073	**0.589**	**1**	**0.564**	**0.743**	**0.750**	**0.645**	-0.095	**0.964**
MIC	**-0.587**	**0.868**	0.061	**0.555**	**0.564**	**1**	**0.608**	**0.755**	**0.757**	-0.008	**0.529**
NOO	**-0.737**	**0.720**	**-0.170**	**0.796**	**0.743**	**0.608**	**1**	**0.831**	**0.786**	**-0.136**	**0.757**
MPC	**-0.716**	**0.857**	-0.002	**0.695**	**0.750**	**0.755**	**0.831**	**1**	**0.872**	-0.087	**0.699**
RFC	**-0.654**	**0.829**	**0.196**	**0.587**	**0.645**	**0.757**	**0.786**	**0.872**	**1**	-0.071	**0.599**
TCC	0.090	-0.068	-0.010	-0.089	-0.095	-0.008	**-0.136**	-0.087	-0.071	**1**	-0.087
WMPC1	**-0.926**	**0.553**	-0.106	**0.589**	**0.964**	**0.529**	**0.757**	**0.699**	**0.599**	-0.087	**1**

Tableau 8:Matrice des corrélations de LUCENE

ORO :

Métrique	Qi	CBO	DOIH	LCOM1	LOC	MIC	NOO	MPC	RFC	TCC	WMPC1
Qi	**1**	**-0.374**	0.108	**-0.727**	**-0.860**	**-0.485**	**-0.763**	**-0.843**	**-0.330**	0.050	**-0.855**
CBO	**-0.374**	**1**	-0.056	0.156	**0.295**	**0.751**	0.175	**0.505**	**0.544**	-0.071	**0.227**
DOIH	0.108	-0.056	**1**	-0.121	-0.075	-0.087	-0.209	-0.036	**0.414**	**0.276**	-0.090
LCOM1	**-0.727**	0.156	-0.121	**1**	**0.751**	**0.303**	**0.874**	**0.702**	**0.236**	-0.076	**0.770**
LOC	**-0.860**	**0.295**	-0.075	**0.751**	**1**	**0.425**	**0.739**	**0.894**	**0.358**	-0.029	**0.979**
MIC	**-0.485**	**0.751**	-0.087	**0.303**	**0.425**	**1**	**0.377**	**0.618**	**0.348**	-0.098	**0.369**
NOO	**-0.763**	0.175	-0.209	**0.874**	**0.739**	**0.377**	**1**	**0.748**	**0.273**	-0.006	**0.763**
MPC	**-0.843**	**0.505**	-0.036	**0.702**	**0.894**	**0.618**	**0.748**	**1**	**0.485**	-0.016	**0.821**
NOOM	**-0.554**	0.025	0.058	**0.371**	**0.495**	0.204	**0.610**	**0.455**	**0.257**	0.173	**0.546**
RFC	**-0.330**	**0.544**	**0.414**	**0.236**	**0.358**	**0.348**	**0.273**	**0.485**	**1**	**0.241**	**0.299**
TCC	0.050	-0.071	**0.276**	-0.076	-0.029	-0.098	-0.006	-0.016	**0.241**	**1**	-0.030
WMPC1	**-0.855**	**0.227**	-0.090	**0.770**	**0.979**	**0.369**	**0.763**	**0.821**	**0.299**	-0.030	**1**

Tableau 9:Matrice des corrélations d'ORO

SNARK :

Variables	Qi	CBO	DOIH	LCOM1	LOC	MIC	NOO	MPC	RFC	TCC	WMPC1
Qi	**1**	**-0.763**	-0.062	**-0.621**	**-0.837**	**-0.611**	**-0.727**	**-0.810**	**-0.756**	-0.057	**-0.848**
CBO	**-0.763**	**1**	0.024	**0.660**	**0.792**	**0.675**	**0.625**	**0.838**	**0.775**	-0.159	**0.729**
DOIH	-0.062	0.024	**1**	-0.070	-0.001	0.171	-0.210	-0.005	0.278	-0.030	-0.041
LCOM1	**-0.621**	**0.660**	-0.070	**1**	**0.666**	**0.463**	**0.659**	**0.654**	**0.499**	-0.180	**0.601**
LOC	**-0.837**	**0.792**	-0.001	**0.666**	**1**	**0.656**	**0.767**	**0.892**	**0.780**	-0.117	**0.901**
MIC	**-0.611**	**0.675**	0.171	**0.463**	**0.656**	**1**	**0.507**	**0.608**	**0.807**	-0.081	**0.618**
NOO	**-0.727**	**0.625**	-0.210	**0.659**	**0.767**	**0.507**	**1**	**0.823**	**0.660**	-0.242	**0.867**
MPC	**-0.810**	**0.838**	-0.005	**0.654**	**0.892**	**0.608**	**0.823**	**1**	**0.829**	-0.170	**0.909**
RFC	**-0.756**	**0.775**	0.278	**0.499**	**0.780**	**0.807**	**0.660**	**0.829**	**1**	-0.097	**0.763**
TCC	-0.057	-0.159	-0.030	-0.180	-0.117	-0.081	-0.242	-0.170	-0.097	**1**	-0.162
WMPC1	**-0.848**	**0.729**	-0.041	**0.601**	**0.901**	**0.618**	**0.867**	**0.909**	**0.763**	-0.162	**1**

Tableau 10: Matrice des corrélations de SNARK

Pour les matrices des corrélations de GNUJSP, ORO, LUCENE, FREECS et SNARK, les observations faites au niveau de JFLEX et JMOL se confirment. A savoir : des corrélations avec DOIH toujours peu significatives, s'expliquant par l'utilisation d'une hiérarchie d'héritage peu profonde dans ces logiciels, avec des profondeurs moyennes de 1.52, 1.40, 1.70, 1.47, et 1.09. Par ailleurs :

- Remarquons aussi la corrélation entre TCC et Qi significative et élevée pour le cas de GNUJSP.

- La corrélation entre RFC et TCC est inexplicable, car elle est significative dans certains projets et de signe changeant : 0.24 pour ORO et -0.58 pour JSP.

- La forte corrélation qui existe entre DOIH et RFC, rend la métrique RFC floue pour l'interprétation. En effet, cette métrique reste trop sensible à DOIH. Comme une grande valeur et une trop petite valeur de DOIH ne sont pas désirables, RFC devient difficile à interpréter, car cette dernière doit être minimisée pour une meilleure testabilité.

5.2.7.2 Variabilité

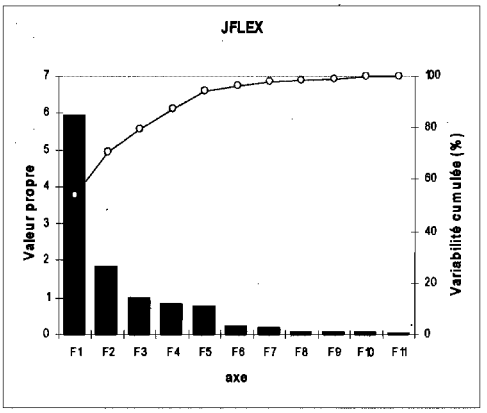


Figure 17:Variabilité des composantes principales JFlex

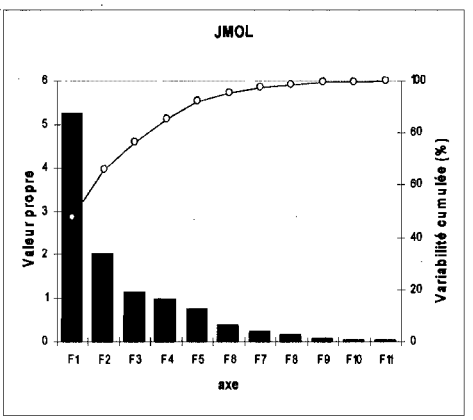


Figure 18:Variabilité des composantes principales JMol

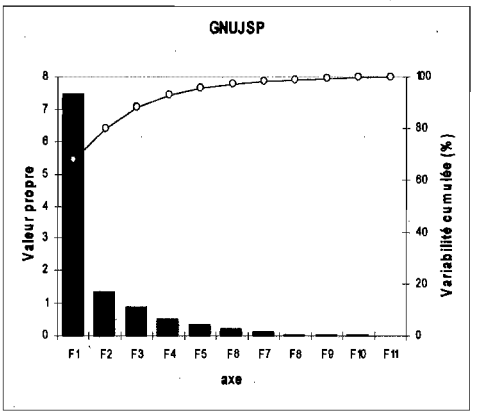


Figure 19:Variabilité des composantes principales Gnujsp

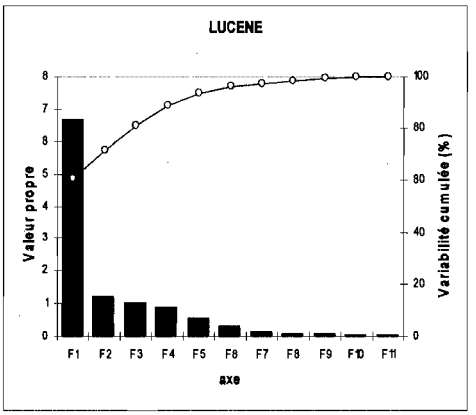


Figure 20:Variabilité des composantes principales Lucen

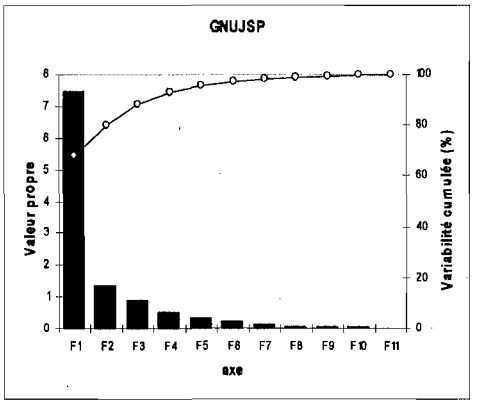


Figure 21:Variabilité des composantes principales Gnujsp

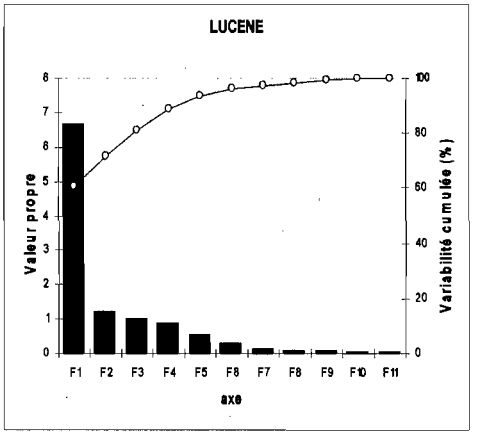


Figure 22:Variabilité des composantes principales Lucen

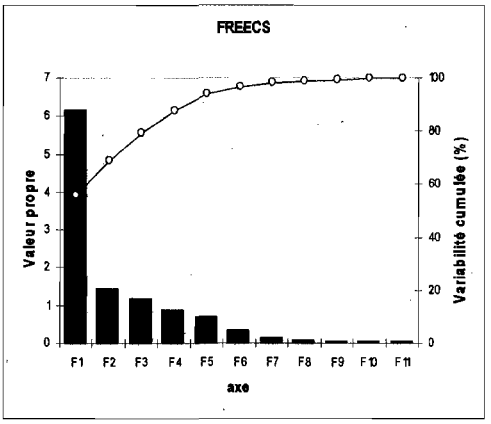


Figure 23:Variabilité des composantes principales Freecs

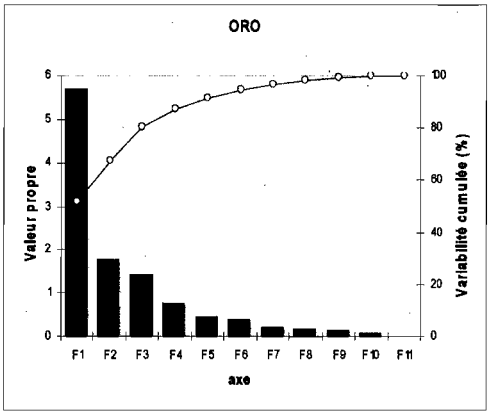


Figure 24:Variabilité des composantes principales Oro

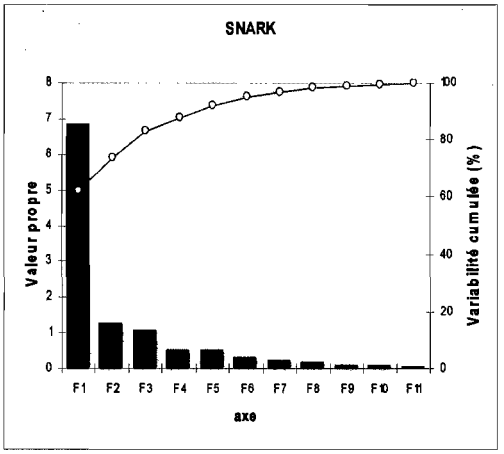


Figure 25:Variabilité des composantes principales Snark.

Pour avoir au moins 95 % de l'information contenue dans les métriques, nous allons retenir les 5 premières composantes principales (F1, F2, F3, F4, F5) pour GNUJSP [figure 21] et les 6 premières pour les autres projets [figures 17; 18; 19; 20; 22; 23; 24]. La distribution de l'information sur les premières composantes est assez uniforme pour les différents projets. Cependant, elle est plus concentrée sur les premières composantes pour GNUJSP. Dans la suite, les autres composantes seront négligées.

En faisant une autre ACP sur le même jeu de données, mais sans l'indicateur de qualité, on voit une diminution de la variabilité de la première composante. Autrement dit, les indicateurs de qualité n'apportent pas seulement de l'information en plus, mais capturent plusieurs métriques, ce qui a pour effet de mieux comprimer l'information sur la première composante. Le tableau 11 donne le pourcentage de contenu informationnel de la première composante (F1) avant et après l'intégration dans l'analyse, des indicateurs de qualité.

	JFLEX	JMOL	FREECS	GNUJSP	LUCENE	ORO	SNARK
Sans Qi	53,944	45,768	55,952	65,879	59,464	48,725	60,821
Avec Qi	53,965	47,742	56,077	67,967	60,627	51,615	62,240
Apport	0,021	1,974	0,125	2,088	1,163	2,890	1,418

Tableau 11:Variabilité de la première composante sans et avec les valeurs des Qi

Contribution des métriques dans les nouvelles composantes

L'analyse par composantes principales nous fournit l'apport de chaque métrique sur les nouvelles composantes obtenues. En nous intéressant aux premières composantes principales et considérant ce qui a été dit plus haut (95% de l'information totale), nous pouvons lister l'ensemble des métriques capturées par les indicateurs de qualité, quantifier (en pourcentage) le taux d'inclusion de ces métriques dans les Qi au niveau de chaque composante et au niveau global des premières composantes.

	F1	F2	F3	F4	F5	F6
Qi	9,951	1,928	0,267	15,250	28,030	0,065
CBO	11,991	5,370	2,142	11,127	2,813	0,304
DOIH	0,070	45,951	0,737	6,813	0,034	11,877
LCOM1	9,148	2,564	2,471	16,220	20,551	32,256
LOC	11,609	1,494	1,209	14,182	14,621	1,613
MIC	10,957	0,108	1,417	25,919	6,230	24,164
MPC	14,617	0,009	0,028	5,140	5,091	2,156
NOO	14,193	0,987	0,266	0,469	9,887	6,039
RFC	2,698	40,176	0,003	1,412	0,506	14,843
TCC	0,678	0,019	91,459	2,176	4,838	0,324
WMPC1	14,091	1,393	0,001	1,292	7,399	6,360

Tableau 12:Contribution des métriques pour JLEX

	F1	F2	F3	F4	F5	F6
Qi	13,727	0,004	2,809	0,377	6,031	28,458
CBO	12,351	7,131	2,822	0,601	10,803	2,094
DOIH	4,253	27,390	16,020	0,210	0,094	1,267
LCOM1	4,138	17,886	24,512	0,782	7,445	14,111
LOC	12,079	0,195	7,286	0,017	22,934	20,699
MIC	8,045	0,000	14,414	1,864	45,938	1,524
MPC	16,325	0,289	3,631	0,308	2,296	12,986
NOO	9,726	15,650	11,977	0,001	0,031	1,804
RFC	6,946	20,488	16,309	0,000	0,028	0,011
TCC	0,590	0,136	0,195	95,828	2,217	0,951
WMPC1	11,819	10,832	0,025	0,012	2,184	16,094

Tableau 13:Contribution des métriques pour JMOL

	F1	F2	F3	F4	F5	F6
Qi	10,017	2,023	6,918	1,011	10,403	51,936
CBO	10,307	10,513	11,419	3,510	1,121	0,700
DOIH	0,025	30,564	17,478	33,110	6,973	0,685
LCOM1	6,572	15,683	7,011	0,000	38,142	2,020
LOC	13,738	0,908	0,095	0,376	12,497	0,035
MIC	5,660	19,465	10,675	12,543	11,417	8,924
NOO	11,903	10,444	1,387	0,202	9,075	0,118
MPC	14,352	0,008	1,218	0,462	3,082	15,652
RFC	13,541	2,555	0,027	4,457	0,281	13,416
TCC	0,019	5,038	43,186	44,046	1,049	6,508
WMPC1	13,865	2,799	0,587	0,282	5,960	0,007

Tableau 14:Contribution des métriques pour FREECS

	F1	F2	F3	F4	F5
Qi	12,076	1,103	3,322	1,130	0,231
CBO	10,604	0,039	12,442	2,706	0,063
DOIH	1,014	57,540	1,109	24,311	0,874
LCOM1	7,271	0,135	27,896	14,303	39,190
LOC	11,506	1,915	5,969	1,411	0,513
MIC	10,098	1,778	13,271	0,039	17,114
NOO	8,642	0,267	27,592	2,191	17,726
MPC	11,658	1,778	6,908	2,620	0,145
RFC	11,543	3,885	0,031	1,975	11,135
TCC	4,182	29,270	0,382	49,204	7,349
WMPC1	11,407	2,289	1,078	0,110	5,660

Tableau 15: Contribution des métriques pour GNUJSP

	F1	F2	F3	F4	F5	F6
Qi	11,343	4,109	0,036	13,467	2,611	0,834
CBO	10,891	6,420	0,447	9,574	8,969	0,046
DOIH	0,019	51,047	6,774	30,100	6,413	0,522
LCOM1	8,766	0,507	0,004	4,505	57,971	17,475
LOC	11,575	5,579	0,044	13,327	1,251	0,234
MIC	9,595	9,097	1,327	5,142	11,184	35,372
NOO	12,276	1,574	0,019	2,357	9,566	12,261
MPC	12,945	1,426	0,036	1,736	0,000	8,077
RFC	11,353	9,985	0,035	0,399	0,044	24,163
TCC	0,188	1,693	91,263	4,803	1,511	0,375
WMPC1	11,049	8,562	0,014	14,591	0,480	0,641

Tableau 16:Contribution des métriques pour LUCENE

	F1	F2	F3	F4	F5	F6
Qi	14,701	0,573	0,114	0,017	4,570	0,460
CBO	4,279	18,641	23,457	1,237	0,562	1,972
DOIH	0,168	21,703	20,171	30,308	4,261	17,230
LCOM1	11,976	4,946	2,597	0,347	18,907	10,182
LOC	15,246	0,885	1,524	0,522	10,503	6,547
MIC	6,657	7,563	20,527	2,043	4,449	31,064
NOO	12,709	4,118	1,619	1,436	20,070	8,334
MPC	15,761	0,470	0,019	0,018	2,107	0,136
RFC	3,981	29,799	1,778	1,803	23,976	18,854
TCC	0,029	9,051	25,651	61,799	1,707	0,501
WMPC1	14,492	2,250	2,542	0,470	8,889	4,721

Tableau 17:Contribution des métriques pour ORO

	F1	F2	F3	F4	F5	F6
Qi	11,577	0,324	5,275	1,765	3,864	0,430
CBO	11,257	0,172	0,023	0,248	10,890	46,563
DOIH	0,014	63,927	7,082	13,124	8,302	0,923
LCOM1	7,946	3,726	0,369	56,355	17,558	7,639
LOC	12,786	0,083	0,347	0,018	1,431	1,089
MIC	8,295	8,426	0,012	19,285	33,416	15,144
NOO	10,620	8,821	0,159	0,236	7,883	20,456
MPC	13,108	0,282	0,001	0,031	3,825	6,070
RFC	11,303	9,136	0,044	6,299	0,057	0,062
TCC	0,424	4,200	86,598	1,226	0,016	0,603
WMPC1	12,671	0,902	0,092	1,412	12,757	1,020

Tableau 18:Contribution des métriques pour SNARK

Les tableaux 12 à 18 montrent la quantité informationnelle apportée par chaque métrique sur les composantes. Nous pouvons calculer sur chaque composante la quantité d'information que contiennent les indicateurs de qualité et que recoupent les autres métriques. En nous inspirant des travaux de K. Aggarwal et al. [Aggarwal 06], nous pouvons associer chaque composante principale à un attribut de qualité.

- F1 est une composante qui regroupe principalement des métriques de taille. Nous associerons F1 à la taille. Les métriques regroupées sont notamment LOC et RFC, cependant d'autres métriques corrélées à la taille et à RFC ont également un apport significatif. C'est le cas de MPC qui est une métrique de couplage, et WMPC une métrique de complexité.

F2 est moins floue que F1 car seule, la métrique DOIH apporte constamment un contenu informationnel important, même si certaines autres métriques dans le cas de certains projets ont un apport non négligeable c'est le cas de RFC pour le projet ORO, JMOL, JFLEX, et de TCC pour GNUJSP. Nous associerons F2 à l'héritage.

- F3 regroupe beaucoup d'information venant de TCC qui est une métrique de cohésion. (86.60% avec SNARK, 25.65% avec ORO, 91.26% avec

LUCENE, 43.17% avec FREECS, 91.46% avec JFLEX). Dans les projets GNUJSP et JMOL, LCOM représente les taux les plus importants, soit respectivement 27.90% et 24.51%. Nous associerons cette colonne à la cohésion.

- F4 associe une métrique de cohésion (TCC) et une métrique d'héritage (DOIH); dans cette composante peuvent parfois intervenir des métriques de couplage par invocation (MIC). Nous ne lui associerons aucun attribut cependant elle reste fortement teintée de cohésion.

F5 et F6 sont plus difficiles à expliquer, vu les contributions irrégulières des différentes métriques dans leurs compositions.

5.2.7.3 Taux de couverture moyenne des indicateurs de qualité

Dans cette section, nous calculons le taux de couverture des autres métriques par les indicateurs de qualité. Nous avons fait la somme des couvertures sur toutes les composantes que nous avons pondérées par l'apport informationnel de la composante. La couverture de Qi sur une métrique pour une composante est donnée par le rapport entre la contribution de Qi et celle de la métrique pour la même composante. Si le rapport est plus grand que 1, on le remplace par 1.

$$TC(Qi/CBO) = \sum_{k=1}^{6} Cont(F_k) \cdot TC_{F_k}(Qi/CBO)$$

$$TC_{F_k}(Qi/CBO) = \begin{cases} 1 & si \ \dfrac{Cont(Qi/F_k)}{Cont(CBO/F_k)} > 1 \\ \dfrac{Cont(Qi/F_k)}{Cont(CBO/F_k)} & sinon \end{cases}$$

Avec : $Cont(F_k)$ le pourcentage de contribution de la k-ieme composante F_k sur l'ensemble et $Cont(Qi/F_k)$, le pourcentage de contribution de l'indicateur de qualité Qi dans la composante F_k . Le tableau 19: donne les résultats obtenus

par cette formule moyenne pondérée pour chaque projet ainsi que la moyenne pour tous les projets.

- Pour CBO : Cette métrique de couplage peut être remplacée par Qi, car celui-ci capture jusqu'à 75.67% de son information. Cela s'explique par la prise en compte du couplage effectif à travers les appels de méthodes (uniquement) par le modèle des indicateurs de qualité. Le reste de l'information non capturée se retrouve dans :

	JFLEX	JMOL	FREECS	GNUJSP	LUCENE	ORO	SNARK	Moyenne
CBO	69.70	70.60	75.06	87.19	80.54	57.16	89.47	75.67%
DOIH	72.49	68.67	70.67	77.30	69.77	56.28	73.34	69.79%
LCOM	81.65	62.11	81.04	81.58	89.21	55.51	73.96	75.01%
LOC	80.22	66.06	80.85	84.55	90.56	63.36	87.53	79.02%
MIC	82.04	74.07	73.69	82.29	75.07	57.08	73.14	73.91%
MPC	77.11	67.37	77.74	83.64	85.77	55.96	79.02	75.23%
NOO	78.17	69.27	80.17	84.34	86.44	91.13	85.24	82.11%
RFC	78.31	68.64	73.51	82.29	87.21	53.68	80.65	74.90%
TCC	85.70	68.61	72.47	76.75	87.29	60.03	74.83	75.10%
WMPC1	78.44	77.04	77.93	86.76	89.98	58.96	77.64	78.11%

Tableau 19:Pourcentage de couverture des Qi sur les différentes métriques

- Le couplage causé par le flux de données; couplage que ne capturent pas les indicateurs de qualité. Notons que GNUJSP et SNARK ont le plus fort taux de capture de CBO par Qi. En effet, GNUJSP et SNARK ont peu de variables publiques (0% pour GNUJSP et 1% pour SNARK), dès lors, tous les échanges de flux entre classes (couplage effectif) se font à travers les appels de méthodes.

- Le couplage avec les librairies systèmes, qui sont aussi ignorés par Qi.

- Pour DOIH : Cette métrique de couplage est en partie (69.8%) capturée par Qi. Cependant, nous avons montré plus haut que Qi ne devenait sensible à DOIH que quand le concept d'héritage est très présent dans le projet. Qi peut

être utilisé à la place de DOIH pour déceler les effets d'un héritage très complexe dans un projet.

- LCOM : Qi n'ayant aucun rapport direct avec la cohésion au sens de LCOM, la bonne couverture de Qi observée sur cette métrique (75.01%) peut s'expliquer par l'intermédiaire d'une métrique de couplage. En admettant qu'une faible cohésion entraîne un fort couplage qui fait baisser le Qi. La relation (négative) qui existe entre le couplage et la cohésion a fait l'objet de travaux récents de Badri et al. [Badri 08].

- LOC : Mesurant la taille d'un système. Il varie dans le même sens que sa complexité, son couplage et certaines de ses autres métriques; ce qui peut expliquer sa couverture de 79.02% par Qi.

- WMPC représente la complexité cyclcomatique. Elle est présente dans le modèle d'indicateur de qualité à travers l'indicateur de fiabilité intrinsèque des méthodes, ce qui explique un taux de capture de 78.11%. L'information sur la complexité restante étant constituée d'arcs de graphe ne menant pas à des appels de méthodes, donc ignorées par notre modèle.

5.2.8 Conclusions sur l'ACP

Cette expérimentation d'envergure a permis de constater que les métriques NOO, CBO, LOC, MIC, MPC, RFC, WMPC et LCOM sont capturées à hauteur de 75% par les Qi. Pour certaines métriques, la capture se fait de manière directe. En effet, WMPC, NOO et CBO sont naturellement incluses dans le modèle Qi alors que la corrélation avec d'autres métriques comme le LCOM ou TCC ne s'explique que transitivement, c'est-à-dire, par l'existence de lien entre la cohésion et une métrique tierce (probablement CBO).

D'un autre côté, les Qi ne capturent le DOIH que si le mécanisme d'héritage est très important dans le système analysé. Enfin, on a remarqué que le critère d'encapsulation est capital pour que les Qi donnent des résultats interprétables.

5.3 Changeabilité

5.3.1 Problématique

« La seule constance c'est le changement ». Cette expression à elle seule, justifie l'intérêt de pouvoir prédire les changements intervenants dans un logiciel. Les enjeux reliés à cette question importante du processus de développement et de maintenance des systèmes logiciels, en particulier ceux qui sont d'envergure et complexes, sont nombreux. Pour saisir l'importance de cet aspect du logiciel, rappelons que plusieurs études ont montré que la maintenance du logiciel coûte cher, de l'ordre de 60 à 75 % de tout l'effort fourni lors de son développement [Sommerville 04, Presman 05]. De ce fait, des efforts importants sont déployés depuis la phase d'analyse et de conception pour tenir compte de l'évolution du logiciel. Durant ces phases, des prospections sur les diagrammes, et les modèles établis permettent de préparer le logiciel à faciliter les changements, et à contenir leurs impacts. Par ailleurs, les efforts fournis par les concepteurs pour augmenter la lisibilité, la testabilité, la réutilisabilité, diminuer le couplage… etc., tant d'éléments qui contribuent de façon directe ou indirecte à faciliter la maintenance, conditionnent le système pour les futurs changements lors de son cycle de vie. L'avènement des changements peut faire suite entre autres à des besoins d'extension du logiciel, ou de mises au point suite à la découverte de fautes. Le bon conditionnement du système devrait avoir pour effet de limiter les propagations des effets du changement. Une autre manière d'assurer ce bon conditionnement logiciel serait de prédire le changement, et pouvoir ainsi, réajuster les structures conceptuelles du logiciel en conséquence. Dans leur modèle, les indicateurs de qualité (Qi) regroupent un certain nombre de métriques telles que le couplage, la complexité. Par ailleurs, grâce aux expérimentations précédentes avec l'ACP,

les indicateurs de qualité capturent d'autres métriques de manière indirecte (comme la cohésion et la taille). Toutes ces métriques sont souvent associées aux changements. Plusieurs travaux de recherche ont abordé des facettes de cette problématique en cherchant par exemple des liens éventuels entre la taille, le couplage et la cohésion (entre autres) et les changements. Ceci nous amère directement à étudier les liens entre les Qi et les changements.

5.3.2 Objectifs

Par ce protocole, nous voulons vérifier expérimentalement que plus l'indicateur de qualité d'une classe est faible, plus elle est susceptible de subir des changements. Pour y arriver, nous étudions le lien qui existe entre les indicateurs de qualité et le nombre de changements que subissent les classes d'une version à l'autre d'un logiciel. Ensuite, pour tenir compte du fait que certaines fautes impliquant des changements peuvent être découvertes après plusieurs versions, nous allons étudier les liens entre les Qi et la somme des changements sur plusieurs versions d'un logiciel.

Deux types d'analyses ont été effectués sur deux logiciels. L'une par la régression logistique, l'autre par l'étude de corrélations entre les Qi et la somme des changements sur plusieurs versions. Ces deux méthodes devraient démontrer expérimentalement la capacité de prédiction des changements des Qi ainsi que la qualité des prédictions. Nous débutons les expérimentations par le tracé du diagramme en moustache (Boxplot) qui donnera une vue globale des tendances entre les valeurs de Qi et les changements.

5.3.3 Démarche

1- Choix des logiciels JFLEX et JMOL 14 et 6 versions successives
2- Pour chaque version (sauf pour la dernière), calcul des Qi des classes,

3- Recherche de changements entre les classes de la version précédente et la version suivante. La valeur du changement est binaire : 1 pour changement, 0 pour non-changement.
4- Tracé des diagrammes en boîte (pour visualiser les tendances)
5- Calcul des corrélations entre les changements et les Qi (1ère confirmation des observations des diagrammes),
6- Étude de régression logistique ($2^{ème}$ confirmation des tendances observées précédemment),
7- Calcul du total des changements dans toutes les versions,
8- Calcul de corrélation entre la moyenne des Qi des n premières versions et le nombre total de changement,
9- Les étapes 6...9 seront répétées pour chacune des métriques décrites plus bas à des fins de comparaison avec les résultats obtenus par les Qi.

5.3.4 Environnement et collecte des données

5.3.4.1 Choix des Systèmes à analyser

Nous avons choisi un grand nombre de versions de deux logiciels open source. Il s'agit de JFlex et de JMol (voir section 5.2.4). Les 13 versions de JFlex à partir de la version 1.2 seront retenues pour cette expérimentation, version durant laquelle JFlex a subi des changements sans changer la structure fondamentale de son diagramme de classe. Le logiciel est passé de 43 à 60 classes. Les changements intervenus sont souvent liés à la correction des fautes, et à l'ajout de fonctionnalités. JMol à subit d'importants changements à partir de sa version 6, de ce fait et pour les tests de corrélations pour la prédiction a long terme, nous nous arrêterons à la version 5. Cependant, les 9 versions seront retenues pour le test de régression logistique, et la visualisation des tendances dans le diagramme en boîte. Les interfaces seront exclues des systèmes.

5.3.4.2 Choix des métriques témoins

5 métriques ont été retenues parmi celles présentées au chapitre 2, pour servir de base de comparaison avec les indicateurs de qualité : LCOM pour la cohésion, DOIH pour l'héritage, CBO pour le couplage de, LOC pour la taille, et WMPC pour la complexité. Ces métriques sont très souvent associées aux changements [Tsantalis 05]. Une classe qui a un faible couplage, ne subit pas beaucoup l'effet de propagation des changements. Dans un code d'une grande complexité, se glissent plus souvent des fautes qui seront découvertes pendant les tests ou l'utilisation du logiciel [Basili 96]. Une forte cohésion témoigne d'une bonne conception, ce qui évite à la classe de subir des ajustements structurels au cours du cycle de vie du logiciel. Les classes de grande taille sont plus sujettes à des fautes, il s'agit juste d'un constat probabiliste. Et enfin l'héritage, quand il est peu utilisé, cela dénote souvent un manque de réutilisation de code. Mais les profonds hiérarchies d'héritages sont synonymes de grande complexité conceptuelle, et structurelle du système, qui nécessite plus d'effort pour la maintenance.

5.3.4.3 Calcul des changements

Pour recueillir les données du changement, nous utilisons le modèle de Tsantalis et al. [Tsantalis 05] qui distingue, deux types de changements dans une classe selon leurs origines :

Les changements internes, qui sont en fait des modifications propres de la classe, pour notamment corriger un bug, améliorer ou optimiser le code; et les changements externes qui résultent de la mise à niveau d'une classe suite a des changements dans d'autres classes (ripple effect – effet de propagation). Nous prendrons en compte tous les changements, qu'ils soient internes ou externes.

L'état d'une classe pour une version donnée sera caractérisé par la variable binaire « change », qui est vraie (1) si la classe a effectivement changée entre la version précédente et la version courante, et faux (0) dans le cas contraire. La

somme de la variable « change » de chaque classe sur N générations, nous donne la fréquence des changements subis sur les versions. Pour certaines expérimentations, notamment sur la prédiction de changement à long terme (sur N générations), nous ne retiendrons que les classes qui n'ont pas été supprimées durant les N générations.

5.3.5 Statistiques générales des systèmes

Nous retrouvons les statistiques de JMOL et JFEX au tableau 2 de la section [5.2.5]. Ce tableau nous donne quelques statistiques de l'ensemble des systèmes que nous avons analysés. Les 24 systèmes ont un total de 350 000 lignes de codes près de 3255 classes et 17821 méthodes. 7.4% de classes de nos systèmes sont contenues dans un arbre d'héritage (soit environ 241classe qui héritent d'une classe mère autre que la classe Object). La présence du système d'héritage permettra de vérifier que nos indicateurs de qualité capturent l'effet du polymorphisme. Le nombre de variables publiques/privées (PVPb et PVPv) reflète le degré d'encapsulation des données. Plus de variables privées rendent le flux de contrôle beaucoup plus important que le flux de données, et donc nos indicateurs de qualité deviennent plus précis avec l'encapsulation.

5.3.6 L'approche statistique

Pour cette expérimentation, nous traçons dans un premier temps, le diagramme en boîte pour visualiser la tendance des prédictions avant d'utiliser la régression logistique, ensuite des études de corrélations seront réalisées pour confirmer et mesurer nos tendances.

5.3.6.1 Diagrammes en boîte

Ces diagrammes nous donnent une tendance générale de la variable « change » par rapport aux Qi des classes correspondantes. Ce sont des représentations uni variées d'échantillons de données simples et assez complètes puisque dans la

version proposée par XLSTAT sont affichés le minimum, le 1er quartile, la médiane, la moyenne, le 3e quartile, ainsi que les deux limites au-delà desquelles on peut considérer que les valeurs sont anormales. La moyenne est affichée sous la forme d'un " + ", et la médiane sous la forme d'une ligne noire. Les limites sont ainsi calculées :

Limite inf : $\mathbf{limInf = X(i_0)\ tel\ que\ X(i_0) = \min_i (Q1 - 1.5 * (Q3 - Qi))}$

Limite sup : $\mathbf{limSup = X(i_0)\ tel\ que\ X(i_0) = \min_i (Q3 + 1.5 * (Q3 - Qi))}$

En dehors de l'intervalle $\mathbf{]Q1 - 3(Q3-Q1)\ ;\ Q3 + 3(Q3-Q1)[}$, les valeurs sont affichées avec un symbole *; et les valeurs comprises dans [Q1 - 3 (Q3 – Q1); Q1 – 1.5 (Q3 – Q1)] ou [Q3 + 1.5 (Q3 – Q1); Q3 + 3 (Q3 – Q1)] sont affichées avec un symbole "o".

5.3.6.2 Régression logistique

La régression logistique est une méthode très utilisée, car elle permet de modéliser des variables binaires ou des sommes de variables binaires. Elle est surtout utilisée dans le domaine médical (guérison ou non d'un patient), en sociologie, en de l'épidémiologie (analyse d'enquêtes), en marketing quantitatif (achat ou non de produits ou services suite à une action) et en finance pour la modélisation de risques (scoring).

Le principe du modèle de la régression logistique consiste à relier l'observation ou la non-observation d'un événement à des variables explicatives. L'utilisation de ce modèle dans notre cas à pour objectif d'évaluer à partir de quelle valeur des indicateurs de qualité, une classe va subir au moins un changement dans la prochaine version du logiciel.

La régression logistique et la régression linéaire appartiennent à la même famille des modèles GLM (Generalized Linear Models): dans les deux cas, on relie un événement à une combinaison linéaire de variables explicatives.

Pour la régression linéaire, la variable dépendante suit une loi normale N(μ,s) où μ est une fonction linéaire des variables explicatives. Pour la régression logistique, la variable dépendante, aussi appelée variable réponse, suit une loi de Bernouilli de paramètre p (p étant la probabilité moyenne pour que l'événement se produise). Lorsque l'expérience est répétée une fois, nous avons une loi Binomiale (n, p) si l'expérience est répétée n fois. Le paramètre de probabilité p correspond à une fonction d'une combinaison linéaire des variables explicatives.

Les fonctions les plus couramment utilisées pour relier la probabilité p aux variables explicatives sont la fonction logistique (on parle alors de modèle Logit) et la fonction de répartition de la loi normale standard (on parle alors de modèle Probit). Ces deux fonctions sont parfaitement symétriques et sigmoïdes (en forme de « S »).

L'expression analytique des modèles est donnée par : Logit : $p = \frac{e^{\beta x}}{1+e^{\beta x}}$ où βx représente la combinaison linéaire des variables (constante comprise).

La connaissance de la loi de distribution de l'événement étudié, permet d'écrire la vraisemblance de l'échantillon. Pour estimer les paramètres β du modèle (les coefficients de la fonction linéaire), on cherche à maximiser la fonction de vraisemblance. Contrairement à la régression linéaire, une solution analytique exacte n'existe pas. Il est donc nécessaire d'utiliser un algorithme itératif. Le logiciel XLSTAT utilise un algorithme de Newton-Raphson, on peut modifier si on le souhaite le nombre maximum d'itérations et le seuil de convergence.

Pour un point de séparation (cutoff) donné, on peut dresser un tableau de classification (aussi appelé matrice de confusion) permettant de calculer un pourcentage d'observations bien classées. Par exemple, pour une valeur de 0.5 du point de séparation, si la probabilité est inférieure à 0.5, l'observation est considérée comme étant affectée à la classe 0, sinon, elle est affectée à la classe 1. La courbe ROC (Receiver Operating Characteristics) permet de visualiser la performance d'un modèle pour toutes les valeurs du point de séparation comprise entre 0 et 1. Les termes utilisés viennent de la théorie de détection du signal. On désigne par sensibilité (sensivity) la proportion d'événements positifs bien classés. La spécificité (specificity) correspond à la proportion d'événements négatifs bien classés. Si l'on fait varier la probabilité seuil (cutoff) à partir de laquelle on considère qu'un événement doit être considéré comme positif, la sensibilité et la spécificité varient. La courbe des points (1-spécificité, sensibilité) est appelée la courbe ROC. L'aire sous la courbe (ou Area Under the Curve – AUC) est un indice synthétique évaluer pour les courbes ROC. L'AUC correspond à la probabilité pour qu'un événement positif ait une probabilité (donnée par le modèle) plus élevé qu'un événement négatif. Pour un modèle idéal, on a AUC=1, pour un modèle aléatoire, on à AUC=0.5. On considère habituellement que le modèle est bon dès lors que la valeur de l'AUC est supérieure ou égale à 0.7. Un modèle bien discriminant doit avoir une AUC entre 0.87 et 0.9. Un modèle ayant une AUC supérieure à 0.9 est excellent.

5.3.6.3 Corrélations

Nous utiliserons le coefficient de corrélation de Pearson. Cette statistique est le coefficient de corrélation le plus communément utilisé. Elle est plus adaptée a nos données quantitatives, malgré qu'elles soient discontinues. En effet, la présence de la variable binaire « change » fait que beaucoup de rang ex-æquo se retrouvent dans nos mesures, ce qui biaise les tests de corrélation basées sur les rangs. La valeur de la corrélation de Pearson est comprise entre -1 et 1, et elle mesure le niveau de relation linéaire entre deux variables. Les p-values

calculées pour les coefficients de corrélation permettent de tester l'hypothèse nulle de corrélation non significativement différente de zéro entre les variables. Cependant, il faut être prudent quant à l'interprétation des corrélations. En effet, si l'indépendance entre deux variables implique la nullité du coefficient de corrélation entre les variables, la réciproque n'est pas vraie : on peut avoir une corrélation proche de zéro entre deux variables parce que la relation n'est pas linéaire, ou parce qu'elle est complexe et nécessite la prise en compte d'autres variables. Le coefficient de corrélation de Spearman utilise les rangs des observations et non leur valeur en tant que telle. Ce coefficient est donc en théorie plus adapté aux données ordinales sans beaucoup d'ex-æquo. Comme pour le coefficient de Pearson, on peut aussi interpréter ce coefficient en termes de variabilité expliquée. Ici, il s'agit bien entendu de la variabilité des rangs.

5.3.7 Résultats et interprétations

5.3.7.1 Diagramme en boîte

Les diagrammes des figures 26 et 27 nous donnent la tendance globale des changements survenus pour chaque classe. Dans les deux cas (JMol et JFlex), les boîtes sont beaucoup plus centrées vers 1 (vers le haut) pour les classes inchangées que pour les classes ayant subi un changement. Ce qui s'interprète ainsi: Globalement, les classes ayant subi un changement ont un Qi plus faible que celles qui sont restées inchangées. Le coefficient de corrélation est de 0.415 pour JFlex et de 0.197 pour JMol. Cependant, la tendance est plus marquée pour JMol que pour JFlex.

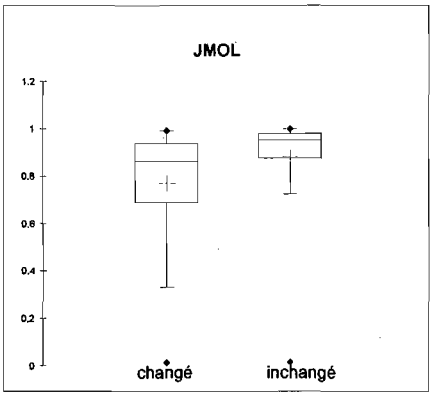


Figure 27: Boxplot JMOL

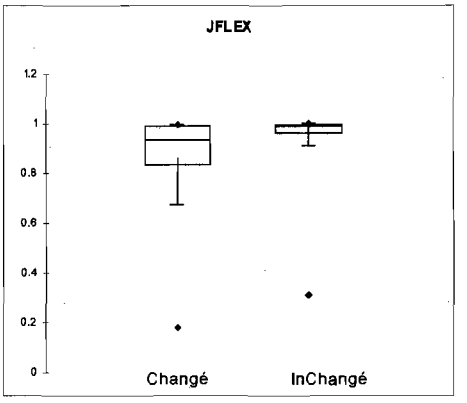


Figure 26: Boxplot JFLEX

Avec une moyenne de 0.773 et 0.832 respectivement pour les classes ayant subi des changements pour JMol et JFlex contre 0.885 et 0.964 pour les classes inchangées, le tableau 20 montre que les classes qui ont de faibles valeurs de Qi ont plus de chance de subir des changements. Ce résultat est confirmé par les valeurs de la médiane, du premier et du troisième quartile.

	JMOL		JFLEX	
Statistique	changé	inchangé	changé	inchangé
Minimum	0.014	0.014	0.175	0.313
Maximum	0.994	1.000	0.997	0.999
Moyenne	0.773	0.885	0.832	0.964
Amplitude	0.980	0.986	0.821	0.686
1er Quartile	0.687	0.876	0.834	0.96
Médiane	0.860	0.951	0.936	0.987
3ème Quartile	0.938	0.977	0.987	0.994

Tableau 20:Valeurs représentées par les Boxplots

5.3.7.2 Régression logistique

Le tableau 21 nous donne les paramètres normalisés du modèle logistique. Pour chaque logiciel, sont affichés l'estimation du paramètre, l'écart-type correspondant, le Khi² de Wald, et les intervalles « profile likelihood ».

L'hypothèse H0 (H0 : Y=p0 :) correspond au modèle indépendant qui donne la probabilité p0 (p0= 0,189 pour JFlex et 0,108 pour JMol) quelques soient les valeurs des variables explicatives; on cherche à vérifier si le modèle ajusté est significativement plus performant que ce modèle. Nous effectuons le test de Wald suivant une loi du Khi² de degrés de liberté 1.

Source	β	Ecart-type	Khi² de Wald	Pr > Khi²
JMOL	-0.243	0.029	71.859	< 0,0001
JFLEX	-0.533	0.086	38.438	< 0,0001

Tableau 21:Coefficients normalisés

Pour les deux logiciels, les probabilités que ces hypothèses se réalisent sont trop faibles (<0.0001), ce qui nous emmène à rejeter le H0. Notre modèle est donc plus performant que le cas aléatoire. Le coefficient bêta (normalisé) est largement au-dessus de zéro, ce qui démontre l'apport non négligeable en information de Qi dans le modèle.

Les graphes des figures 28 et 29 montrent les courbes de ROC de JMol et JFlex, qui se démarquent bien de la première bissectrice. Elles nous montrent l'évolution de la spécificité et la sensibilité en fonction du point de séparation (cutoff).

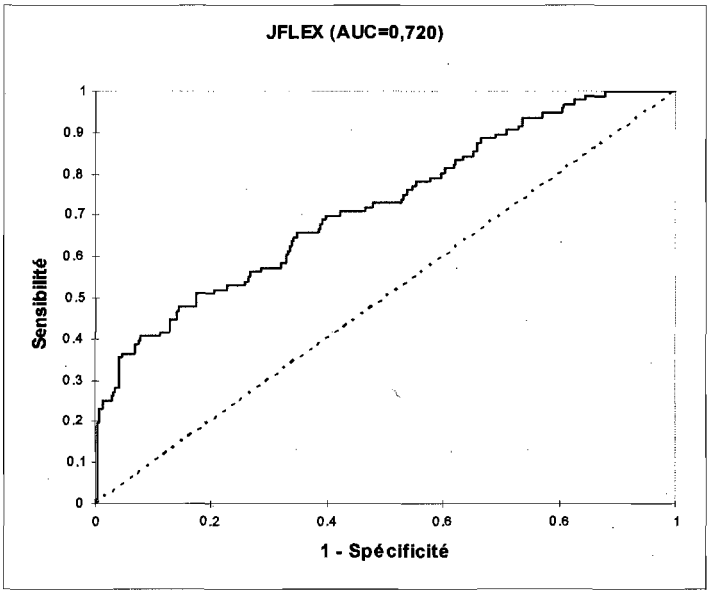


Figure 28: JFlex ROC (AUC=0.720)

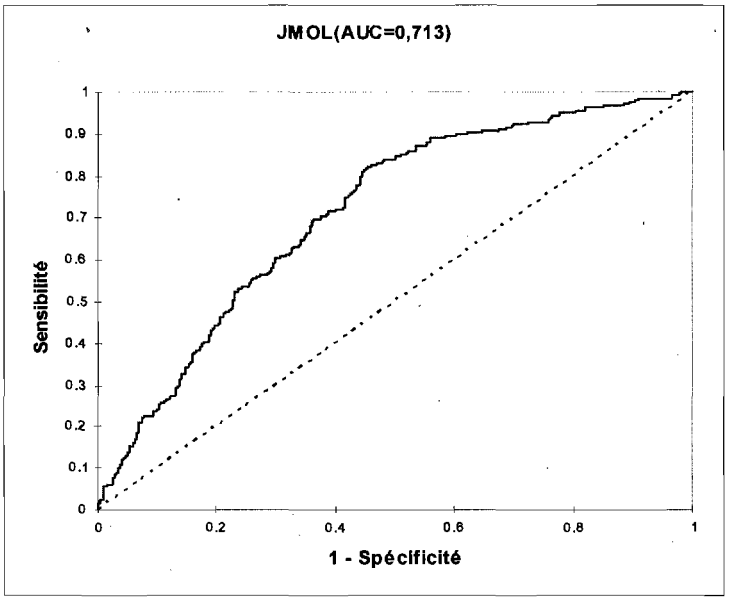


Figure 29: JMol ROC (AUC=0.713)

La meilleure performance est obtenue pour une spécificité égale à 0.55 et une sensibilité égale à 0.8 pour JMol, tandis que pour JFlex, une spécificité égale à 075, et une sensibilité égale à 0.55 donnent les meilleures classifications pour ce modèle. L'AUC est le seul indicateur qui tient compte de la performance globale du modèle (pour tous les cutoff). Il est de 0.720 pour JFlex et 0.713 pour JMol, ce qui permet de conclure que notre modèle est un bon discriminant.

	JFLEX	JMOL
Qi	0.720	0.713
CBO	0,68	0.657
DOIH	0,49	0.550
LCOM	0,582	0.679
LOC	0,696	0.718
WMPC	0,67	0.721

Tableau 22: AUC pour les courbes ROC des autres métriques

Le tableau 13 montre les valeurs d'AUC des différentes métriques témoins. Pour JFlex, Qi représente le meilleur modèle logistique pour prédire les

changements, tandis que pour JMol, WMPC a la meilleure valeur.0.72. Qi peut être qualifié de bon discriminant car c'est la seule métrique à dépasser le seuil de 0.70 dans les deux systèmes.

5.3.7.3 Prédiction des changements à long terme

La prédiction des changements à long terme entre une version i et une version k>i, permet de tenir compte (si le système ne subit pas de réajustement global depuis la version i) de l'aspect probabiliste de la découverte des bugs dans les systèmes logiciels. Dans le tableau 23, les indicateurs de qualité, ainsi que la plupart des métriques voient leur corrélation avec la fréquence des changements augmenter au cours des versions.

Pour les Qi, la corrélation avec les changements s'explique par la complexité qu'ils capturent. L'augmentation constante de sa valeur (de 0.668 à 0.710 en valeur absolue) au cours des versions découle naturellement du calcul probabiliste intégré dans le modèle. En effet, nous convenons qu'un module très usité voit ses fautes résiduelles découvertes plus tôt qu'un module peu appelé. Le signe négatif de cette corrélation signifie que la diminution des Qi s'accompagne d'une augmentation de la fréquence des changements.

Pour le CBO, la corrélation avec les changements pourrait s'expliquer par le constat suivant: une classe fortement couplée subit plus l'impact du changement. L'augmentation de cette corrélation (de 0.546 à 0.581), même si elle n'est pas constante, reste difficile à interpréter.

DOIH ne présente pas de corrélation avec le changement, JMol utilise peu (par rapport à JFlex) le mécanisme d'héritage. Notons que la structure d'héritages a aussi peu évolué pour JMol d'où une quasi-constance de la trace de corrélation avec le changement.

RFC ne présente pas de grande corrélation avec les changements (autour de 0.14). De plus, elle reste constante durant les 8 versions de JMol.

	v 1	v 1.1	v 1.2	v 2	v 3	v 4	v 5	v 6
Qi	-0,660	-0,660	-0,684	-0,696	-0,696	-0,708	-0,709	-0,716
CBO	0,546	0,546	0,546	0,557	0,562	0,567	0,568	0,581
DOIH	0,060	0,060	0,060	0,060	0,060	0,063	0,056	0,061
LCOM1	0,498	0,499	0,500	0,501	0,495	0,495	0,498	0,529
LOC	0,609	0,640	0,640	0,648	0,651	0,656	0,680	0,696
NOO	0,624	0,625	0,625	0,629	0,628	0,625	0,625	0,626
RFC	0,148	0,148	0,148	0,151	0,153	0,154	0,148	0,147
WMPC	0,689	0,689	0,690	0,690	0,690	0,698	0,706	0,710

Tableau 23:Evolution des corrélations JMOL avec les versions

LCOM, NOO présentent une corrélation avec les changements conformément au constat que nous avions fait plus haut (section sur le choix des métriques témoins). Cette corrélation n'évolue pas au cours des versions. En effet, la cohésion et la taille (nombre de méthodes) ne présentent aucun aspect probabiliste.

LOC présente des corrélations avec les changements comme beaucoup d'études l'ont montré. La croissance de celle-ci avec les versions, reste toute fois difficilement interprétable. Les classes de grande taille semblent être plus utilisées.

Enfin, WMPC, qui mesure la complexité présente la deuxième plus forte corrélation, ce qui est conforme aux études menées antérieurement et qui lient la complexité aux fautes, donc aux changements [Basili 84]. Cependant, elle croit moins vite que celle de Qi, car le modèle WPMC n'est pas probabiliste.

L'augmentation de corrélation de WMPC et LOC avec les versions du logiciel reste, cependant, très difficile à interpréter.

Pour interpréter les résultats de JFlex [tableau 25], nous avons besoin de rappeler les informations statistiques concernant le flux de contrôle et le flux de

données. Le manque d'encapsulation de JFlex par rapport à JMol peut expliquer grandement les plus faibles corrélations retrouvées pour la prédiction des changements à long terme de JFlex. En effet, le flou introduit par une moyenne de 20 % de variables publiques sur les 14 versions analysées (flux de donnée potentiellement important, au dépend du flux de contrôle) contre 12% pour les 6 versions de JMol, fait qu'une grande partie de l'impact des modifications remonte le flux de données (ignorés totalement par les Qi) au détriment du flux de contrôle. Une pondération dans le calcul de Qi par la complexité cyclomatique (Qipdr dans le tableau 24) améliore grandement les résultats. Cependant, et encore une fois, l'aspect probabiliste des Qi explique la plus forte croissance des corrélations de ces derniers, soit une variation de 0.075. Le signe de cette corrélation conserve tout son sens, en conformité avec les hypothèses. L'utilisation de l'héritage par JFlex (plus que JMol) fait apparaître des corrélations significatives avec le changement. Les autres métriques ont gardé les mêmes tendances de résultats que pour JMol, et RFC confirme même (de manière expérimentale) son indépendance vis-à-vis des changements.

	V1.2	V1.2.1	V1.2.2	V1.3	V1.3.1	V1.3.2	V1.3.3	V1.3.4	V1.3.5	V1.4p1	V1.4p3	V1.4p4	V1.4p5	delta
Qipdr	-0,795	-0,794	-0,791	-0,796	-0,798	-0,800	-0,799	-0,799	-0,799	-0,800	-0,801	-0,808	-0,812	-0,017
Qi	-0,573	-0,574	-0,604	-0,616	-0,622	-0,624	-0,627	-0,629	-0,630	-0,632	-0,633	-0,642	-0,648	-0,075
CBO	0,695	0,696	0,697	0,712	0,720	0,725	0,736	0,743	0,748	0,752	0,756	0,761	0,764	0,069
DOIH	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	-0,267	0,000
LOC	0,801	0,801	0,801	0,800	0,801	0,801	0,800	0,800	0,799	0,801	0,802	0,803	0,804	0,004
NOO	0,586	0,587	0,580	0,592	0,601	0,606	0,614	0,620	0,624	0,628	0,632	0,639	0,644	0,059
RFC	0,090	0,093	0,095	0,103	0,110	0,116	0,123	0,129	0,133	0,136	0,140	0,144	0,148	0,059
WMPC	0,601	0,601	0,598	0,606	0,613	0,617	0,622	0,627	0,630	0,634	0,638	0,642	0,645	0,044

Tableau 24:Évolution des corrélations pour JFlex, delta =Corrélation (v1.2) - Corrélation (v1.4p5)

5.3.8 Conclusion sur la changeabilité

Cette deuxième série d'expérimentations a permis de montrer la capacité des indicateurs de qualité (Qi) à prédire les changements par deux approches différentes. L'une focalisant sur les changements d'une version a l'autre par génération d'un modèle logistique probabiliste, l'autre permettant de suivre l'évolution d'une classe sur plusieurs versions d'un logiciel. Les deux approches ont produit des résultats concurrençant ceux des métriques traditionnelles. Nous avons aussi mis à jour certains critères clefs permettant de faire de la bonne prédiction avec ce modèle. Il s'agit de l'encapsulation. Une étude plus approfondie sur le type de changement (adaptatif ou correctif), au niveau plus granulaire (méthode) pourrait donner des résultats meilleurs. Cette dimension sera retenue dans nos travaux futurs.

5.4 Testabilité

Le processus de test est une étape clés du processus de développement du logiciel. Son objectif est de découvrir le maximum d'erreurs introduites lors des étapes d'analyse, de conception et d'implémentation. La testabilité d'un logiciel est un attribut de qualité qui caractérise essentiellement la facilité de réaliser le test d'un logiciel.

5.4.1 Problématique

L'importance de l'étape de test repose sur le principe que plus tôt une faute et découverte, moins cher en termes de coût, et d'efforts, elle sera résorbée [Presman 05]. En effet, on imagine aisément de nos jours, les effets d'une faute sur des logiciels critiques, pilotant des systèmes dans des domaines aussi diversifiés que sensibles que sont la santé, les transports, l'aérospatiale, ou encore les systèmes bancaires. Il est alors important de réaliser des batteries de tests les plus exhaustives possibles. Cependant, en raison de l'explosion combinatoire des chemins de flux dans un système logiciel, il serait judicieux et

raisonnable de revoir à la baisse, les ambitions des concepteurs de logiciel souhaitant couvrir l'intégralité des chemins de test par une équipe de développeurs limitée en termes de ressources humaines et économiques. Une alternative serait de garder à l'esprit, le long du processus, la testabilité.

Autrement dit, intégrer dans notre démarche le souci de préparer les produits en gestation aux tests, ce qui nécessite des outils et des indicateurs permettant de prédire l'effort nécessaire pour tester les différents modules de notre logiciel. Les indicateurs de qualité peuvent s'inscrire à ce niveau, au même titre que certaines métriques et permettre à l'équipe de développement de faire des choix éclairés, qui faciliteront les tests, dans les phases de développement et de maintenance (conception des tests, orientation des tests, couvertures, ...).

5.4.2 Objectifs

Ce protocole a pour but de mesurer la testabilité des classes grâce aux Qi; Il nous permet de voir aussi si les Qi sont meilleurs en tant qu'indicateur de testabilité que les autres métriques définies dans la littérature. Il consiste en une étude de corrélation entre les mesures de taille des classes-tests et d'autres métriques pour les classes logicielles correspondantes incluant les Qi. Les classes-test sont les classes moniteurs ou driver réalisées par les programmeurs pour tester une classe du système. Elles peuvent être générées en utilisant des outils supports tels JUnit.

5.4.3 Démarche

1- Choix d'un logiciel ainsi que ses classes-test.
2- Calcul des Qi et d'autres métriques (choisies au préalable) des classes des logiciels
3- Calcul des métriques (taille, complexité... etc.) de toutes les classes classes-test

4- Calcul des corrélations de Spearman entre les métriques de classes et celles des classes-tests.
5- Détermination des métriques de testabilité.

5.4.4 Environnement et collecte des données

La batterie de test que l'on analysera sera celle réalisée par les développeurs du logiciel grâce à JUnit; nous supposerons (à juste titre) que l'effort que ces derniers mettent dans l'écriture de la classe test d'une classe donnée (testabilité) est proportionnel à la taille de celle-ci et à sa complexité; autrement dit, nous assimilerons la testabilité d'une classe à la taille et à la complexité de la classe-test qu'elle nécessite.

5.4.4.1 Choix des systèmes à analyser

Pour cette dernière série d'expérimentations, notre choix s'est tourné vers un logiciel développé par le processus XP dans lequel, la rigueur des tests est constante et très soutenue. Le logiciel Ant choisi, est développé comme un sous-projet du serveur Web apache; son code source comporte environ 170 000 lignes de code java, il a 887 classes contenues dans 87 paquetages, inspirés par l'approche expérimentale de M. Bruntink et al. [Bruntink 04]. L'objectif consiste à mesurer la testabilité des classes logicielles qui ont des classes-test. Elles sont au nombre de 116 pour la version 1.5.3-1 d'Ant.. Les programmeurs ont implémenté les cas de test JUnit durant le développement, sans critère particulier.

5.4.4.2 Métriques choisies

Pour les classes test, nous mesurerons leur complexité par leur complexité cyclomatique, (TestWMPC) leur taille par le nombre de lignes de code (TestLOC). Pour les classes système, nous avons choisi les métriques qui ont fourni les meilleurs résultats pour les expérimentations de M. Bruntink et

al. [Bruntink 04], à savoir : LOC, WMPC, NOM et LCOM, auxquelles nous ajouterons MIC, et les Qi. (Voir chapitre 2 et 5.2.4 pour la présentation de ces métriques). Toutes les métriques exposées ici sont calculées par l'outil Borland Together 2007. Dans la suite, les valeurs de corrélations calculées sont celles de Spearman.

5.4.5 Résultats et interprétations

Dans le tableau 25, toutes les valeurs des corrélations sont significatives au sens alpha =0.05

Variables	TestWMPC	TestLOC
Qi	-0,397	-0,591
CBO	0,389	0,369
FO	0,378	0,383
LCOM1	0,380	0,476
LOC	0,439	0,543
MIC	0,324	0,343
NOO	0,428	0,539
WMPC2	0,423	0,544

Tableau 25: Corrélation de Spearman

Les résultats de ce tableau 25 nous enseignent qu'en supposant que l'effort de test d'une classe donnée est proportionnel (a juste titre) a la complexité de sa classe-test, (mesurée par WPMC), alors la métrique LOC prédit mieux la testabilité que les autres métriques (0.439 de corrélation). D'un autre côté, en ne considérant que la dimension taille de la classe-test, Qi donne la meilleure prédiction pour la testabilité avec 0.591 de corrélation.

5.4.6 Conclusion sur la testabilité

Nous avons expérimenté notre métrique en tant qu'indicateur de testabilité. Les résultats montrent que les Qi pourraient donner de meilleurs résultats que plusieurs autres métriques pour prédire la testabilité d'une classe logicielle. Cependant, la comparaison avec d'autres modèles de mesure de testabilité et la diversification des systèmes analysés pourraient bien confirmer de manière

définitive que Qi est une métrique de testabilité. Ceci fera l'objet de travaux futurs.

Chapitre 6

Conclusions

La métrologie logicielle reste le moyen le plus objectif pour mesurer la qualité d'un système logiciel, malgré la diversité des métriques, et le manque de validation de certaines qu'on peut trouver dans la littérature. Le modèle d'indicateurs de qualité que nous avons présenté dans ce mémoire ouvre de nouvelles perspectives quant à l'utilisation des probabilités pour unifier les métriques et capturer de manière plus concise certains attributs de qualité très complexes de haut niveau. Ces métriques sont très utiles lors de la maintenance des applications. Les expérimentations que nous avons réalisées et présentées dans les chapitres précédents montrent une précision inégalée sur la mesure des différents attributs de la qualité du logiciel.

Avec plus de 60% du prix du logiciel, dans de nombreux cas, la maintenance constitue une phase continue et constante d'efforts de correction de fautes et de réadaptation du logiciel aux besoins du client. Il s'agit de supporter cette phase sans altérer la qualité des applications. Ce défi justifie largement l'attention que lui porte la recherche en génie logiciel. Les indicateurs de qualité s'inscrivent dans ce cadre. Ils ont, au vu des expériences menées et présentées dans ce mémoire, apporté une nette amélioration par rapport aux autres métriques dans la prédiction des changements et de la testabilité. Par ailleurs, les indicateurs de qualité peuvent se substituer à la plupart des métriques de base définies au chapitre 2, comme nous l'a démontré par l'utilisation de l'analyse par composantes principales. Des expérimentations plus poussées et plus ciblées dans le cadre de la maintenance devraient permettre de valider définitivement cette métrique.

L'outil développé pour calculer les indicateurs de qualité peut aussi servir à analyser l'impact de changement, à détecter les couplages cycles et

éventuellement de leur maillon faible. Ces études feront l'objet de travaux futurs. La validation de ces dernières propriétés fera des Qi une agrégation de la plus part des métriques de maintenabilité, et un support très puissant pour la maintenance logicielle.

Bibliographie

[Abreu 96] F. B.; Melo, W. L. "Evaluating the Impact of Object-Oriented Design on Software Quality", in the Proceedings of 3rd International Software Metrics Symposium (Metrics'96), Berlin, Germany, March 96

[Abreu95] Abreu, F. Brito; Goulão, Miguel and Esteves, Rita, "Toward the Design Quality Evaluation of Object-Oriented Software Systems".in the Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, Oct. 95

[Aggarwal 06] K.K. Aggarwal, S. Yogesh, K Arvinder, M. Ruchika "Empirical study of Object-Oriented Metrics" Journal of Object Technologie vol 5 No 8 Nov/Dec 2006

[Badri 95-a] L. Badri, M. Badri et S. Ferdenache: "Towards Quality Control Metrics for Object-Oriented Systems Analysis", in the Proceedings of *TOOLS Europe'95* (*Technology of Object-Oriented Languages and Systems*), Versailles, France, Editions Prentice-Hall International, Mars 1995.

[Badri 95-b] M. Badri, L. Badri et S. Layachi : « Vers une stratégie de tests unitaires et d'intégration des classes dans les applications orientées objet », in Revue *Génie Logiciel*, Numéro 38, Paris, Décembre 1995.

[Badri 03] L. Badri & M. Badri: "A New Approach for Class Cohesion Assessment: An empirical validation on several systems", Proceedings of the 7th *International Conference on Software Engineering and Applications*, SEA'2003, Marina del Rey, CA, USA, November 2003 (***Best Paper Award***).

[Badri 04] L. Badri & M. Badri: "A Proposal of a New Class Cohesion Criterion: An Empirical Study", in *Journal of Object Technology* (JOT), vol. 3, no. 4,pp. 145-159, Special issue **(best papers)** of TOOLS USA 2003, April 2004.

[Badri 05] L. Badri, M. Badri & D. St-Yves: "Supporting Predictive Change Impact Analysis: A Control Call Graph Based Technique", in Proceedings of the 12th *Asia-Pasific Software Engineering Conference* (APSEC 2005), *IEEE Computer Society Press*, Taipei, Taiwan, Décembre 2005.

[Badri 08] L. Badri, M. Badri & Alioune Gueye: "Revisiting Class Cohesion: An empirical investigation on several systems", in *Journal of Object Technology* (JOT), Juillet – Août 2008, à paraitre.

[Basili 96] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," IEEE Transactions on Software Engineering. vol. 22, no. 10, pp. 751-761, Oct. 1996.

[Basili 84] Basili, V. R. and Perricone, B. T. 1984. "Software errors and complexity: an empirical investigation". *Commun. ACM* 27, (Jan. 1984), 42-52.

[Bieman 95] J.M. Bieman, B.K. Kang. "Cohesion and Reuse in an Object-Oriented System", in the Proceedings of ACM Symposium on Software Reusability. (SSR '94), pp. 259-262, Seattle, WA April 1995.

[Briand 99] L.C. Briand, J. Wüst, and H. Lounis. "Using Coupling Measurement for Impact Analysis in Object-Oriented Systems", in the Proceedings of IEEE, International Conference Software Maintenance (ICSM), pp. 475-482, Oxford, England, Aug./Sept. 1999.

[Bruntink 04] M. Bruntink, A. Van Deursen "Predicting class testability using object-oriented metrics" in Proceedings of the Fourth IEEE international Workshop on Code source analysis in Manipulation 2004

[Chidamber 94] S.R. Chidamber, C.F. Kemerer. "A metrics Suite for Object Oriented Design". IEEE Transactions on Software Engineering, Vol.20, No.6, June 1994.

[Churcher 95] N.I. Churcher and M.J. Shepperd, "Comments on "A Metrics Suite for Object-Oriented Design", IEEE Trans. Software Engineering, vol. 21, no. 3, pp. 263-265, Mar. 1995.

[El Emam 01] El Emam, K.; Benlarbi, S.; Goel, N. and Rai, S. N., "The confounding effect of class size on the validity of object

oriented metrics," IEEE Transactions on Software Engineering, 27(7): 630-650, 2001.

[El Emam 99] K. El Emam, S.Benlarbi, Ni. Goel "The Confounding Effect of Class Size on the Validity of Object-oriented Metrics"

[Henderson 96] B.Henderson-Sellers. "Object-Oriented Metrics, Measures of Complexity". edited by Prentice-Hall, 1996.

[Hitz 96] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion In Object-Oriented Systems", Symp. Applied Corporate Computing (ISACC "95), Monterrey, Mexico, Oct.25-27, 1995.

[Li 93] W. Li & S Henry, "Maintenance Metrics for Object-Oriented Paradigm", in Proceedings of the First International Software Metrics Symp., pp. 52-60, May 1993.

[Manna 93] M. Manna "Maintenance burden begging for a Remedy", Datamation, April 1993, pp. 53-63

[McCabe 94] Thomas J. McCabe, Arthur H. Watson, "Software Complexity." Crosstalk, Journal of Defense Software Engineering 7, 12 (December 1994): 5-9.

[Rosenberg 98] L. Rosenberg, T. Hammer, J. Shaw "Software Metrics and Reliability" 9th International Symposium Nov 1998.

[Sommerville] Ian Sommerville, "Software ingeneering" seventh edition British library p492

[Tsantalis 05] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides "Prediction the probability of change in Object-Oriented Systems." IEEE Trans. Software Eng., vol. 31, no. 7, July. 2005.